Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

SIGEML: Sistema Inteligente de Gestão Energética apoiado por Machine Learning

Autores: João Robson Santos Martins
Vinicius Ferreira Bernardo de Lima
Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF
2021

João Robson Santos Martins
Vinicius Ferreira Bernardo de Lima

SIGEML: Sistema Inteligente de Gestão Energética apoiado por Machine Learning

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF

2021

João Robson Santos Martins
Vinicius Ferreira Bernardo de Lima
SIGEML: Sistema Inteligente de Gestão Energética apoiado por Machine Learning/ João Robson Santos Martins
Vinicius Ferreira Bernardo de Lima. – Brasília, DF, 2021-
100 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2021.

1. Machine Learning. 2. Séries Temporais. 3. Curvas de carga. 4. Regressão. I. Prof. Dr. Renato Coral Sampaio. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. SIGEML: Sistema Inteligente de Gestão Energética apoiado por Machine Learning

CDU 02:141:005.6

João Robson Santos Martins
Vinicius Ferreira Bernardo de Lima

SIGEML: Sistema Inteligente de Gestão Energética apoiado por Machine Learning

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 18 de novembro de 2021:

Prof. Dr. Renato Coral Sampaio
Orientador

Prof. Dr. Fábio Macêdo Mendes
Convidado 1

Prof. Dr. Alex Reis
Convidado 2

Brasília, DF
2021

Agradecimentos

Agradeço aos meus pais, Marta e Robson, pelo apoio e amor infinito que me permitiram chegar até aqui.

À minha irmã, Maria Luísa, pela força e maturidade que me inspiram.

À dinda, pelo carinho e cuidado de todos esses anos.

A meus avós, tios, primos, familiares e todas as pessoas que de alguma forma, com palavras ou ações, me influenciaram e me fizeram estar onde estou.

Aos colegas de faculdade, principalmente ao Vinicius, que compartilha a autoria deste trabalho comigo e é um parceiro para a vida, e ao Ícaro, que representa um grande exemplo de serenidade e talento para mim. Foi um árduo caminho, mas sem vocês seria muito mais difícil (e menos divertido).

Aos professores que tive pelo caminho, em especial ao professor Renato Coral, pela orientação na elaboração deste trabalho e por todos os conhecimentos passados ao longo do curso.

E à minha namorada e melhor amiga, Maria Eduarda. Obrigado pela paciência, pelo amor e por ter mudado tudo.

João Robson Santos Martins

Agradeço à minha mãe, Marlene, e ao meu pai, Antônio, por todo amor, carinho e suporte que me fizeram chegar tão longe.

À minha irmã Evily que, mais do que qualquer pessoa, me apoiou em todos os sentidos a conseguir trilhar a árdua jornada até aqui.

À minha irmã Letícia que, por meio de sua didática e paciência, me ajudou a descobrir meu potencial na área da ciências exatas.

Ao meu irmão Reinaldo que sempre me incentivou a ser aplicado nos estudos e deu todo suporte que podia oferecer.

Ao meu irmão Antônio por todos os conselhos práticos que me ajudaram a tomar boas decisões.

Ao meu irmão Ruan pelos momentos de descontração que tornaram a jornada mais leve.

Aos meus colegas e amigos de faculdade, principalmente o João e o Ícaro, por todos os momentos especiais que lembrarei com carinho (incluindo virar noites fazendo trabalho). Mais um ciclo termina na vida, mas espero ter a oportunidade de trabalhar novamente ao lado de pessoas tão estimadas.

A todos os professores que me inspiraram a cursar engenharia de software, principalmente o Prof. Dr. Renato Coral Sampaio que, através de muitas aulas instrutivas e experiências contadas, me reforçaram os motivos da minha escolha profissional.

Mas, acima de tudo, agradeço à Jeová Deus pelo dom da vida e por todos os presentes que me dão felicidade, satisfação e objetivo.

Vinicius Ferreira Bernardo de Lima

"Mas a vereda dos justos é como a luz da aurora,
que vai brilhando mais e mais até ser dia perfeito."
(Bíblia Sagrada, Provérbios 4, 18)

Resumo

Softwares de gestão de energia são instrumentos importantes no auxílio do monitoramento e controle de sistemas elétricos. Um exemplo dessas ferramentas é o SIGE, responsável pelo armazenamento de dados de medidores e pelo suporte à gerência energética de diversos edifícios da Universidade de Brasília. O SIGE, no entanto, não fornece funcionalidades que permitam análises preditivas ou que gerem curvas de carga características do consumo de energia nos prédios dos campi. Desse modo, propõe-se a criação do SIGEML, uma plataforma que visa preencher tais lacunas, por meio de um ambiente que permita a experimentação e treinamento de modelos de *Machine Learning* através de dados coletados e disponibilizados pelo SIGE. Portanto, este trabalho elenca os requisitos gerais do SIGEML e analisa a viabilidade do uso de modelos preditivos integrados a ele, através de experimentos que testam a performance de dois algoritmos (ARIMA e LSTM) para séries temporais e três para curvas de carga (regressão linear, *SVR* e *XGBoost*) em uma amostra de dados coletada pelo SIGE.

Palavras-chaves: Machine Learning. Curvas de carga. Séries temporais. Regressão.

Abstract

Energy management systems are essential tools in helping to monitor and control electrical systems. An example of these tools is SIGE, responsible for storing measurer data and supporting energy management in several buildings at the University of Brasília. However, SIGE does not offer functions allowing predictive analyses or that generate load curves that are characteristic of energy consumption in the campus buildings. Thus, we propose the creation of SIGEML, a platform that aims to fill such gaps, through an environment that allows the experimentation and training of *Machine Learning* models using data collected and made available by SIGE. Therefore, this work lists the general requirements of SIGEML and analyzes the feasibility of using predictive models integrated into it, performing experiments that test the performance of two algorithms (ARIMA and LSTM) for time series and three for load curves (linear regression , *SVR* and *XGBoost*) in a sample of data collected by SIGE.

Key-words: Machine Learning. Load curves. Time series. Regression.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SIGE	Sistema de Gestão Energética
UnB	Universidade de Brasília
API	*Application Programming Interface*
ML	*Machine Learning*
ARIMA	*Autoregressive Integrated Moving Average*
RNN	*Recurrent Neural Network*
LSTM	*Long Short Term Memory*
CPD	Centro de Processamento de Dados
MQTT	*Message Queue Telemetry Transport*
UI	*User Interface*

Sumário

1 Introdução

O Sistema de Gestão Energética (SIGE) da Universidade de Brasília (ANGARITA et al., 2020) desempenha um excelente papel monitorando, em tempo real, informações como consumo de energia, nível de tensão, corrente elétrica e fator de potência de edificações da UnB. Essas informações são apresentadas ao usuário da plataforma de maneira intuitiva e de fácil visualização e navegação.

Entretanto, identificar anomalias nos padrões de consumo energético é uma tarefa complexa de ser realizada por um ser humano. Dessa maneira, este trabalho propõe a construção de um *software* que seja capaz de prever a demanda energética facilitando a detecção de tais anomalias.

Nesse sentido, o objetivo é criar um sistema que, através dos dados coletados por medidores de energia, consiga realizar previsões de consumo. Tal sistema deve ser independente, podendo ser acessado por outros serviços de gestão energética, incluindo o próprio SIGE, e conseguir se readaptar periodicamente ou sob demanda, mantendo predições atualizadas e precisas.

1.1 Justificativa

Como explicado por ANGARITA et al., há um uso cada vez mais comum de recursos computacionais para monitorar e analisar dados que auxiliem a gestão de energia, consequência da tendência em interligar processos e da modernização na indústria. Com isso, surgem sistemas específicos, capazes de coletar, tratar, consolidar e exibir relatórios e indicadores baseados nos dados gerados por sensores e medidores de energia, como é o caso do SIGE da UnB.

No entanto, esse sistema, apesar de já contar com diversas funcionalidades, ainda não permite gerar análises preditivas, como a previsão de curvas de carga ou do consumo esperado para os próximos minutos ou horas. Dessa forma, o sistema não pode, por exemplo, determinar em tempo real e de maneira automática se um valor de consumo se trata de uma anomalia, o que pode estar sendo causado por algum aparelho ligado desnecessariamente ou falha na rede elétrica.

Sendo assim, enxerga-se uma oportunidade de estender as funções do SIGE, por meio do desenvolvimento de um sistema que permita a realização de análises preditivas com base no consumo de energia já capturado e armazenado pelo SIGE.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral a construção de um sistema que integra modelos de *machine learning* capazes de realizar previsões de consumo de energia elétrica considerando o aspecto temporal presente nos dados históricos gerados pelo SIGE. Por meio dele, será possível a realização de análises exploratórias de curvas de carga e a previsão em tempo real do consumo de energia esperado para um determinado período de tempo.

1.2.2 Objetivos Específicos

Com base no objetivo geral descrito na Seção 1.2.1, espera-se alcançar as seguintes metas após a conclusão deste trabalho:

1. Explorar e analisar os dados históricos gerados e armazenados na plataforma SIGE, entendendo a fundo seus padrões sazonais e respondendo questionamentos acerca de sua relação intrínseca com a dimensão temporal;
2. Estudar opções de modelos de *machine learning* que melhor se adaptam a natureza dos dados de consumo coletados;
3. Realizar experimentos avaliando e comparando a performance entre as opções de modelos de *machine learning* encontrados;
4. Definir requisitos gerais do sistema independente a ser desenvolvido;
5. Desenvolver o sistema.

1.3 Organização do Documento

Este documento é organizado nos seguintes capítulos:

1. **Fundamentação Teórica**: apresenta os conceitos utilizados neste trabalho com base em materiais aceitos pela comunidade acadêmica;
2. **Metodologia**: descreve as etapas adotadas para realização dos experimentos e desenvolvimento do sistema;
3. **Experimentos Iniciais**: contém os resultados alcançados em cada etapa de experimentos descrito no capítulo da metodologia;

4. **SIGEML**: aborda os requisitos, a metodologia de desenvolvimento, a arquitetura e os resultados do sistema implementado, nomeado de SIGEML;

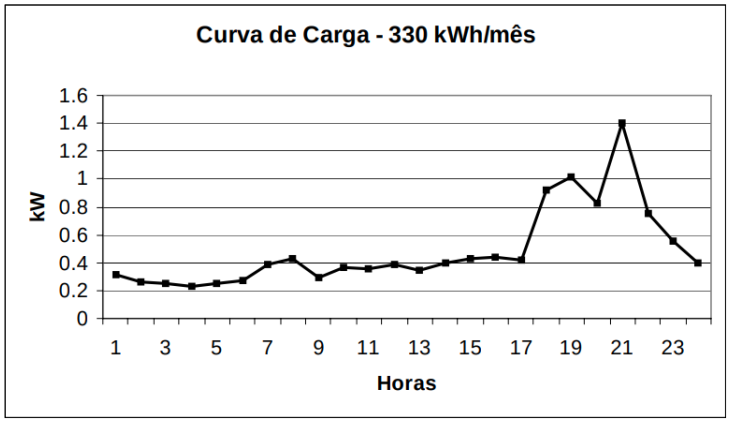
5. **Considerações finais**: apresenta uma análise acerca dos resultados alcançados e mostra possíveis melhorias para trabalhos futuros.

2 Fundamentação Teórica

2.1 Curvas de carga de energia

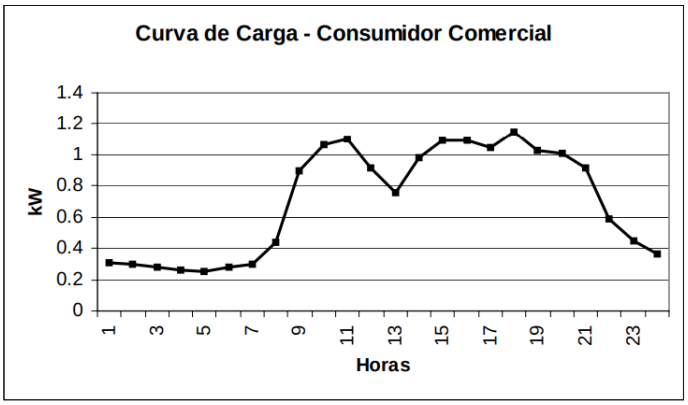
As curvas de carga de um sistema elétrico são representações gráficas da demanda de energia do referido sistema ao longo de um determinado período de tempo, como mostrado nas Figuras 1, 2 e 3. Tais representações são úteis por permitirem uma visualização intuitiva dos padrões de consumo de energia elétrica para diferentes regiões ou até mesmo consumidores específicos. Além disso, possibilitam, até certo grau de precisão, prever com antecedência a demanda e geração de energia necessários para que empresas do setor elétrico forneçam um serviço ininterrupto e de qualidade (ENGELSDORFF, 2019).

Figura 1 – Exemplo de curva de carga para um consumidor residencial



Fonte: (FRANCISQUINI, 2006)

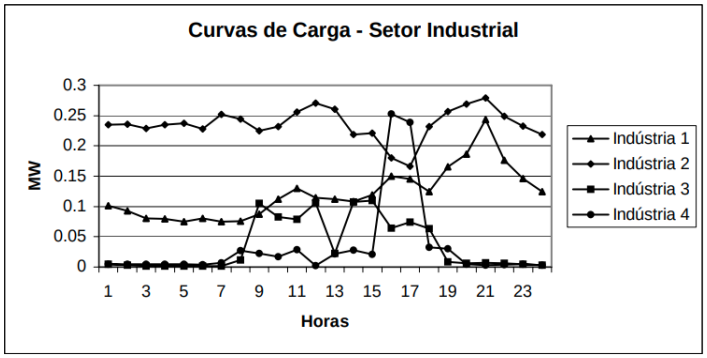
Figura 2 – Exemplo de curva de carga para um consumidor comercial



Fonte: (FRANCISQUINI, 2006)

Evidentemente, inúmeros aspectos e variáveis influenciam a estrutura desses perfis de consumo, em maior ou menor grau, o que pode levar a geração de curvas de carga bastante distintas. Como exemplo para esses fatores, tem-se: desenvolvimento econômico,

Figura 3 – Exemplo de curva de carga para múltiplas indústrias

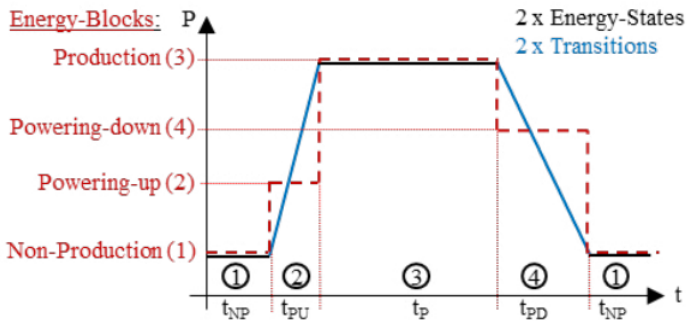


Fonte: (FRANCISQUINI, 2006)

estações climáticas, épocas festivas do ano além da distinção entre dias úteis e fins de semana. (ENGELSDORFF, 2019)

No entanto, mesmo curvas de carga de contexto distintos ainda levam fatores semelhantes entre si, sendo geralmente possível particioná-las em dois estados de consumo de energia e dois períodos de transições entre os estados, como mostra a Figura 4.

Figura 4 – Ilustração dos possíveis estados de consumo de energia e suas transições



Fonte: (FLICK et al., 2020)

Essas etapas, de maneira geral, podem ser detalhadas em (FLICK et al., 2020):

1. *Non Production*: estado de consumo energético com níveis mínimos de demanda. Seu tempo de duração t_{NP} e início variam de acordo com o contexto;

2. *Powering up*: período de transição entre o estado *Non Production* e *Production*. Seu tempo de duração t_{PU} indica o tempo necessário para sair de um patamar de demanda energética mínima para máxima;

3. *Production*: estado de demanda de energia máxima durante o período de tempo t_P. Tal período pode variar de acordo com cada consumidor específico e suas características únicas que influenciam no consumo;

4. *Powering down*: período de transição entre o estado *Production* e *Non Production*. O período t_{NP} indica o tempo decorrido necessário para que o ciclo se reinicie com os sitema voltando ao primeiro estado de consumo.

Assim como os exemplos de sistemas elétricos consumidores apresentados, a Universidade de Brasília, objeto de estudo deste trabalho, possui características únicas e condições externas específicas que influenciam a demanda de energia e consequentemente a elaboração de previsões do consumo. Nesse caso particular da universidade, há, por exemplo, a presença de prédios com objetivos de utilização diferentes, como edifícios com laboratórios, salas de aula, computadores servidores, administrativos e assim por diante.

Além disso, os fatores sazonais fortemente presentes nas atividades realizadas na Universidade podem auxiliar na captura de padrões e elaboração de curvas de carga. No entanto, devido a pandemia do COVID-19, a UnB vive um período atípico no qual as atividades de ensino presenciais estão suspensas desde o dia 23 de março de 2020.

Dessa maneira, é fundamental considerar, na construção de um sistema de geração automática de perfis de curva de carga, modelos de *machine learning* capazes de generalizar e diferenciar diferentes períodos e condições de consumo, como o período excepcional de atividades remotas encarado pela Universidade de Brasília.

2.2 Sistema de gestão energética - SIGE

Softwares de gestão de energia são ferramentas que podem ser utilizadas por equipes de gestão para consolidar e tratar dados coletados via medidores ou centrais de dados. Dessa maneira, essas ferramentas possibilitam a geração de relatórios de consumo e a criação de indicadores que permitem a automatização do monitoramento e otimização do trabalho, assim como a geração de alertas de irregularidades em tempo real (ANGARITA et al., 2020).

Nesse contexto, foi desenvolvido o SIGE da UnB, que tem o foco de aperfeiçoar a gerência do monitoramento energético da instituição como um todo, expandindo "o conhecimento de características de consumo da instalação"e possibilitando o "desenvolvimento de ações pontuais de eficiência energética"(ANGARITA et al., 2020).

Dessa forma, como apresentado por ANGARITA et al., a partir da "coleta periódica de grandezas elétricas instantâneas como tensão, corrente, consumo, geração, fator de potência, potência ativa/reativa/aparente"e de "grandezas agregadas como o consumo e geração de energia por períodos"e do armazenamento de todos esses valores para consulta posterior, o sistema possibilita, por exemplo, a geração de gráficos e relatórios baseados nos dados coletados e a exportação desses dados em formato bruto, além da geração de notificações sobre o mau uso do sistema elétrico.

2.3 *Machine Learning*

Machine Learning, ou aprendizado de máquina, em português, refere-se a "projetar algoritmos que automaticamente extraem informações valiosas de dados"(DEISENROTH; FAISAL; ONG, 2020). Para identificar e assimilar esses padrões em um conjunto de dados, é necessário que esses algoritmos (ou modelos, como são usualmente chamados) aprendam.

Modelos são, de maneira geral, funções matemáticas que, dado uma entrada, geram saídas específicas. Dessa forma, "ensinar", no contexto de *Machine Learning*, trata-se do processo de otimizar os parâmetros de uma função de modo que ela seja capaz de identificar da melhor maneira possível os padrões existentes no dado.

Comumente, esse aprendizado é dividido em dois tipos: supervisionado e não-supervisionado (SHALEV-SHWARTZ; BEN-DAVID, 2014).

O primeiro se refere ao caso em que, para cada amostra presente no dado, há um rótulo ou *label* atribuído. O objetivo do modelo, nesse caso, é aprender a mapear, para cada amostra, da forma mais precisa o possível, entradas compostas de n variáveis ou *features* em saídas que representam os rótulos. Um exemplo desse tipo de aprendizado é a detecção de e-mails que são *spams* e *não-spams* utilizando um conjunto de textos de e-mails com suas categorias previamente assinaladas.

Já o segundo tipo trata da situação onde o dado não contém rótulos. Assim, o objetivo é identificar padrões que permitem associar as amostras de acordo com a similaridade entre suas *features*. Um exemplo disso é agrupar por temas os e-mails do exemplo anterior utilizando apenas seu conteúdo textual.

2.3.1 Regressão

Dentro do contexto de aprendizado supervisionado, existe uma classe de algoritmos que permite realizar o que é chamado de regressão, que tem seu funcionamento fundado em dois componentes principais (CHATTERJEE; SIMONOFF, 2012):

1. Uma variável contínua que se deseja entender ou modelar, geralmente representada por y. Essa variável é chamada de *target*, resposta ou variável dependente e pode representar, por exemplo, o preço de venda de uma casa;

2. Um conjunto de p variáveis que podem ser úteis para prever ou modelar a variável dependente, chamadas de variáveis de previsão ou independentes. Para o exemplo da casa, podem ser representadas pela a área da casa, o número de quartos, etc.

Entretanto, apesar de se basear em dois fundamentos simples, as relações matemáticas entre variáveis dependentes e independente(s) podem não ser, matematicamente

falando, tão triviais. Por conta disso, diversos métodos e abordagens capazes de modelar esse tipo de problema surgiram ao longo dos anos, sendo alguns dos mais utilizados e formalmente citados pela literatura apresentados na seção 2.3.1.1.

2.3.1.1 Abordagens e modelos de regressão

Como explicado anteriormente, há diversas maneiras de modelar problemas de regressão. Neste trabalho, quatro técnicas com fundamentos distintos serão discutidas e aplicadas aos experimentos:

1. Regressão linear;
2. Vetores de suporte;
3. *Gradient Boosting*.

2.3.1.1.1 Regressão linear

Na formulação mais comum de regressão, baseada em situações onde o conjunto de amostras guarda uma relação linear, é possível encontrar uma função na forma

$$y_i = \beta_0 + \beta_1 x_{1i} + ... + \beta_p x_{pi} + \varepsilon_i \tag{2.1}$$

onde os coeficientes β são parâmetros desconhecidos e ε é um termo que representa um erro aleatório (CHATTERJEE; SIMONOFF, 2012).

Dessa maneira, o objetivo primário desse modelo, chamado de regressão linear, é estimar os parâmetros desconhecidos β. Isso deve ocorrer de forma que haja uma representação útil da relação entre o conjunto de variáveis dependentes e independentes, sendo necessário um critério que permita encontrar uma estimativa razoável para esses parâmetros, a fim de minimizar os erros ε.

A abordagem padrão utilizada para esse fim, como explicado por CHATTERJEE; SIMONOFF, se chama método dos mínimos quadrados, onde os valores de β são escolhidos para minimizar

$$\sum_{i=1}^{n} [y_i - (\beta_0 + \beta_i x_{1i} + ... + \beta_p x_{pi})]^2 \tag{2.2}$$

Essa equação, que é representada pela soma das diferenças entre cada valor observado e estimado para y_i (também chamadas de resíduos), permite encontrar o erro total do modelo estimado.

No entanto, é importante observar que essa metodologia considera que os erros para cada par de y observado e predito não são relacionados (CHATTERJEE; SIMONOFF,

2012). Isso quer dizer que se é conhecido que o valor predito para um determinado y_i foi subestimado, por exemplo, não é possível concluir nada sobre o quanto que o modelo irá errar na predição de y_{i+1} ou qualquer outro y.

A violação dessa condição ocorre frequentemente em situações onde se tem uma ordenação cronológica do dado. Tal sequência é geralmente chamada de série temporal. Nessas circunstâncias, erros de observações que são próximas no tempo são frequentemente similares e essa correlação entre elas é chamada de autocorrelação (CHATTERJEE; SIMONOFF, 2012).

Dessa maneira, segundo CHATTERJEE; SIMONOFF, ignorar o relacionamento temporal entre as amostras do dado e consequentemente assumir que não há autocorrelação entre os seus erros pode levar a avaliações enganosas ao se usar uma regressão. Portanto, para um dado que se trata de uma série temporal, utilizar regressão pode não ser a melhor escolha. Nesses casos, deve-se buscar o uso de modelos que lidem com o caráter sequencial das observações, como é demonstrado na seção 2.3.2.

2.3.1.1.2 Máquina de vetores de suporte

Uma máquina de vetores de suporte ou *support vector machine*, em inglês, consiste em uma técnica de aprendizado supervisionado, originalmente concebida para classificação, que tem o objetivo de, dado um conjunto de dados linearmente separável, determinar "um hiperplano que maximize a margem entre amostras positivas e negativas"(DEISENROTH; FAISAL; ONG, 2020). Como essas margens são determinadas por amostras presentes no próprio dado de treino e cada amostra se trata de um ponto n-dimensional no espaço, os pontos mais próximos das margens e consequentemente do hiperplano são conhecidos como "vetores de suporte".

Nesse contexto, como demonstrado por DEISENROTH; FAISAL; ONG, considerando amostras $x_i \in X$ e um hiperplano representado por

$$\vec{w} \cdot x_i + b \tag{2.3}$$

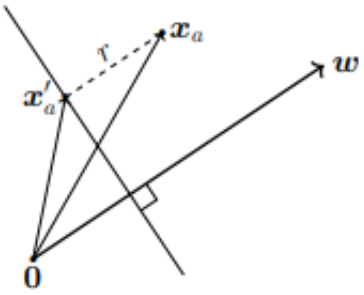
e, tomando como base o objetivo de manter amostras positivas e negativas com *labels* $y_i \in \{1, -1\}$ a distâncias mínimas de 1 e -1 do hiperplano, respectivamente, pode-se definir que

$$y_i(\vec{w} \cdot x_i + b) \geq 1 \tag{2.4}$$

Além disso, sem perda de generalidade, dado que uma amostra x_a está do lado positivo do hiperplano quando $\vec{w} \cdot x_i + b > 0$ e considerando que x_a está a uma distância r do hiperplano, como mostrado na Figura 5, é possível formular que

$$x_a = x'_a + r\frac{w}{\|w\|} \tag{2.5}$$

Figura 5 – Representação do vetor x_a a uma distância r de w



Fonte: (DEISENROTH; FAISAL; ONG, 2020)

Agora, presumindo que a amostra x_a está localizada exatamente na fronteira da margem, ou seja,

$$\vec{w} \cdot x_a + b = 1 \tag{2.6}$$

e levando em conta que sua projeção ortogonal x'_a toca no hiperplano, isto é,

$$\vec{w} \cdot x'_a + b = 0 \tag{2.7}$$

pode-se substituir 2.5 em 2.7, obtendo r:

$$\begin{gathered} \vec{w} \cdot (x_a - r\frac{w}{\|w\|}) + b = \vec{w} \cdot x_a + b - r\frac{w \cdot w}{\|w\|} = 1 - r\frac{\|w\|^2}{\|w\|} = 0 \\ r = \frac{1}{\|w\|} \end{gathered} \tag{2.8}$$

Por fim, maximizando o valor de r, combinando o objetivo de maximizar o tamanho das margens ao mesmo tempo em que se mantém as amostras do lado correto do hiperplano, tem-se

$$\begin{aligned} \max_{w,b} \quad & \frac{1}{\|w\|} \\ \text{sujeito a} \quad & y_i(\vec{w} \cdot x_i + b) \geq 1 \qquad \text{para todo i = 1, 2, ..., N} \end{aligned} \tag{2.9}$$

A fim de tornar essa definição mais conveniente matematicamente (WINSTON, 2010), é possível reescrevê-la como sendo

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{sujeito a} \quad & y_i(\vec{w} \cdot x_i + b) \geq 1 \qquad \text{para todo i = 1, 2, ..., N} \end{aligned} \tag{2.10}$$

Vale ressaltar que nem todo conjunto de dados a ser classificado será linearmente separável e, nessas situações, poderá haver amostras do lado errado do hiperplano ou dentro das margens. Por conta disso, é necessário adicionar uma variável ξ para representar os desvios que ocorrem quando uma amostra x_i se encontra nestas situações, assim como um parâmetro C, responsável por controlar o balanço entre o tamanho da margem e a tolerância permitida para classificações erradas. Com isso, a nova formulação do objetivo é dada por

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n & \\ \text{sujeito a} \quad & y_n(\vec{w}\cdot x_n + b) \geq 1 - \xi_n & \text{para todo n = 1, 2, ..., N} \\ & \xi_n \geq 0 & \text{para todo n = 1, 2, ..., N} \end{aligned} \tag{2.11}$$

Nesse contexto, observa-se que a dimensão de w e consequente a quantidade de parâmetros a ser aprendida é diretamente proporcional ao número de *features* presentes nas amostras. Sendo assim, é importante ressaltar que é possível reescrever a formulação apresentada na Equação 2.11 a partir da utilização do método dos multiplicadores lagrangianos. Por meio dessa técnica, que permite maximizar ou minimizar uma função genérica $f(x, y, z)$ sujeita a uma restrição na forma $g(x, y, z) = k$ (STEWART, 2012) e em que o número de parâmetros cresce proporcionalmente ao número de amostras e não de *features*, o problema de otimização, tomando como base os multiplicadores α e γ, se torna

$$\mathcal{L}(w, b, \xi, \alpha, \gamma) = \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}\alpha_n(y_n(\vec{w}\cdot x_n + b) - 1 + \xi_n) - \sum_{n=1}^{N}\gamma_n\xi_n \tag{2.12}$$

Diferenciando $\mathcal{L}$ em relação a w, b e ξ, obtém-se

$$\begin{aligned} \frac{\partial\mathcal{L}}{\partial w} &= w^T - \sum_{n=1}^{N}\alpha_n y_n x_n^T \\ \frac{\partial\mathcal{L}}{\partial b} &= -\sum_{n=1}^{N}\alpha_n y_n \\ \frac{\partial\mathcal{L}}{\partial \xi_n} &= C - \alpha_n - \gamma_n \end{aligned} \tag{2.13}$$

Igualando as equações em 2.13 a 0, a fim de encontrar os seus máximos, e substituindo os resultados em 2.12, tem-se

$$\mathcal{D}(\xi, \alpha, \gamma) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) - \sum_{i=1}^{N}\alpha_i \tag{2.14}$$

o que resulta no objetivo

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) - \sum_{i=1}^{N} \alpha_i \\ \text{sujeito a} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ 0 \geq \alpha_i \geq C \quad & \text{para todo n = 1, 2, ..., N} \end{aligned} \tag{2.15}$$

Em adição ao uso da variável ξ e das formulações para $\mathcal{L}$, também é necessário e essencial, em determinados casos, o uso de funções conhecidas como *kernels*. Por meio delas, é possível mapear amostras anteriormente não separáveis linearmente pelo hiperplano em um espaço onde essa divisão é possível. A partir disso, considerando uma função *kernel* K, pode-se reescrever 2.15 como

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i \\ \text{sujeito a} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ 0 \geq \alpha_i \geq C \quad & \text{para todo n = 1, 2, ..., N} \end{aligned} \tag{2.16}$$

Um exemplo de função utilizada para esse propósito é a *Radial Basis Function* (RBF), que é dada por (HSU; CHANG; LIN, 2003)

$$K_{rbf}(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0 \tag{2.17}$$

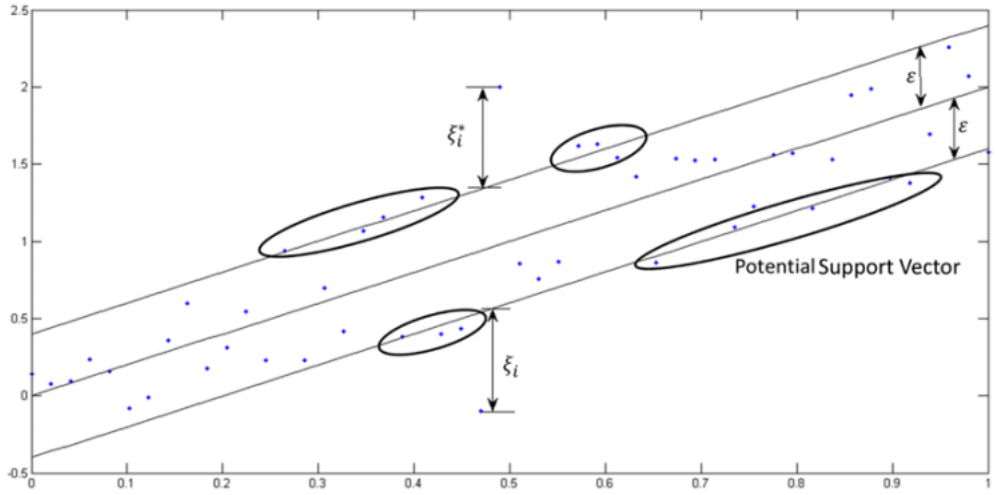
Com isso, tomando como base a Equação 2.16, é possível estabelecer uma nova formulação capaz de realizar regressão utilizando os vetores de suporte (SVR). A ideia, neste caso, é adicionar penalizações, representadas por ξ_i e ξ_i^*, para amostras que estão fora das margens inferiores e superiores aceitáveis, respectivamente. Essas regiões são determinadas por ϵ, como mostrado na Figura 6.

Assim, o novo problema de otimização é dado por (AWAD, 2015)

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N} \xi_n + \xi_n^* & \\ \text{sujeito a} \quad & y_n - \vec{w} \cdot x_n \leq \epsilon + \xi_n^* & \text{para todo n = 1, 2, ..., N} \\ & \vec{w} \cdot x_n - y_n + \leq \epsilon + \xi_n & \text{para todo n = 1, 2, ..., N} \\ & \xi_n, \xi_n^* \geq 0 & \text{para todo n = 1, 2, ..., N} \end{aligned} \tag{2.18}$$

Similar ao que foi feito em 2.12, pode-se aplicar multiplicadores de Lagrange (λ, λ^*, α e α^*) à formulação apresentada em 2.18, obtendo

Figura 6 – Representação de um SVR linear



Fonte: (AWAD, 2015)

$$\begin{aligned}\mathcal{L}(w,\xi,\xi^*,\lambda,\lambda^*,\alpha,\alpha^*) = \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n + \xi_n^* + \sum_{n=1}^{N}\alpha_n^*(y_n - \vec{w}\cdot x_n - \epsilon - \xi_n^*) \\ + \sum_{n=1}^{N}\alpha_n(-y_n + \vec{w}\cdot x_n - \epsilon - \xi_n) - \sum_{n=1}^{N}\alpha_n\xi_n + \alpha_n^*\xi_n^*\end{aligned} \tag{2.19}$$

Por fim, tomando as derivadas parciais das variáveis de $\mathcal{L}$, igualando-as a 0 e realizando as devidas substituições em 2.19, o que inclui a aplicação da função *kernel* no produto interno de x_n e x_j, chega-se em

$$\begin{aligned}\max_{\alpha,\alpha^*} \quad & -\epsilon\sum_{n=1}^{N}(\alpha_n + \alpha_n^*) + \sum_{n=1}^{N}(\alpha_n^* - \alpha_n)y_n - \frac{1}{2}\sum_{j=1}^{N}\sum_{n=1}^{N}(\alpha_n^* - \alpha_n)(\alpha_j^* - \alpha_j)K(x_n, x_j) \\ \text{sujeito a} \quad & 0 \geq \alpha_i, \alpha_i^* \geq C \\ & \sum_{i=1}^{N}(\alpha_n^* - \alpha_n) = 0\end{aligned} \tag{2.20}$$

Como temos na diferenciação que w é

$$w = \sum_{n=1}^{N}(\alpha_n^* - \alpha_n)x_n \tag{2.21}$$

Temos uma função estimada de regressão no seguinte formato:

$$f(x) = \sum_{n=1}^{N}(\alpha_n^* - \alpha_n)K(x_n, x) \tag{2.22}$$

2.3.1.1.3 *Gradient Boosting*

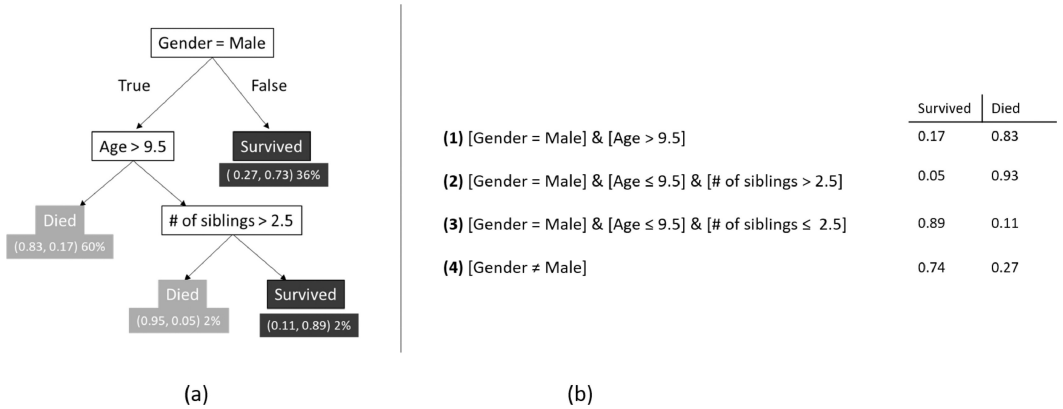
Gradient Boosting, de maneira básica, "consiste num conjunto aditivo de modelos de predição fracos, geralmente árvores de decisão, que otimizam uma função custo diferenciável arbitrária"(KONSTANTINOV; UTKIN, 2021). Portanto, a cada iteração desse algoritmo, novos preditores são criados, buscando fazer com que as predições da variável de interesse a partir das *features*, tomando como base uma função custo arbitrária, sejam gradativamente mais precisas.

Nesse contexto, é possível dividir esses modelos em três componentes principais:

Árvores de decisão: assim como o *gradient boosting*, árvores de decisão ou *decision trees*, em inglês, tratam-se de um método de aprendizado de máquina supervisionado utilizado tanto para regressão quanto para classificação e que é "representado como um conjunto de testes condicionais, onde cada teste compara um atributo com um limite numérico ou um conjunto de valores possíveis"(SAGI; ROKACH, 2020).

Um exemplo dessa estratégia é apresentado na Figura 7, onde é ilustrado o funcionamento de uma árvore de decisão criada a partir de um *dataset* com *features* e *labels* de pessoas que sobreviveriam ou morreriam no naufrágio do Titanic.

Figura 7 – Ilustração de uma árvore de decisão



Fonte: (SAGI; ROKACH, 2020)

Como mostrado, os nós internos da árvore de decisão contém testes condicionais para atributos do *dataset* do naufrágio do *Titanic*. As folhas da árvore contém a classe ou a predição feita para um novo conjunto de *features* a ser classificado. Vale notar que uma das vantagens claras das árvores de decisão é sua fácil interpretação e construção, que contribui para problemas onde é necessário entender como a decisão foi tomada pelo modelo preditivo (SAGI; ROKACH, 2020).

Gradiente descendente: é uma estratégia iterativa que permite encontrar um ponto máximo ou mínimo de qualquer função diferenciável. CHRISTENSEN define o gradiente de uma função diferenciável $f(x)$ como

$$\nabla f(x) = f^{'}(x) \tag{2.23}$$

Dessa maneira, o valor mínimo de uma função, no caso de problemas de minimização, pode ser obtido se a direção de busca se mover na direção negativa do vetor do gradiente. Sendo assim, o algoritmo de minimização de uma função $f(x)$ inicia-se através da escolha arbitrária de um ponto de partida x_0. Em seguida, é calculado o vetor que indica a direção de busca do ponto que minimiza a função através do gradiente $-\nabla f(x_0)$. Esse processo se repete, atualizando o novo ponto de busca a cada etapa, até que o mínimo, local ou global, ou o número de iterações seja atingido. O algoritmo de minimização através do gradiente descendente pode ser resumido como

$$x_{i+1} = x_i - sf_i \nabla f(x_i) \mid 0 <= i <= N \tag{2.24}$$

onde sf_i é um fator de escala de busca ajustado para cada iteração e N o número total de iterações (CHRISTENSEN, 2015).

Função custo: na matemática, uma função custo "é uma função que mapeia um evento ou valores de uma ou mais variáveis em um número real que representa intuitivamente algum 'custo' associado ao evento"(RASCHKA, 2019). Dessa maneira, uma função custo $L(y_i, f(x_i))$ calcula o residual entre uma observação y_i e o valor aproximado da função f no mesmo ponto x_i.

Através da definição dos três principais componentes, o algoritmo do *gradient boosting* pode ser definido nas seguintes etapas (KONSTANTINOV; UTKIN, 2021):

1. **Entrada**: tomando como entrada do algoritmo um conjunto de dados $D\{(x_i, y_i)\}_{i=1}^{n}$ onde n é o total de amostras usadas no treinamento, uma função custo diferenciável $L(y_i, F(x_i))$ e o número total de iterações do algoritmo T;

2. Calcule uma função inicial $F_0(x)$ como um valor constante de maneira que:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) \tag{2.25}$$

3. para todo $m = 1$ até T faça:

   3.1. Calcule o residual r_{im} através da equação:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \mid i = 1, ..., n \tag{2.26}$$

   3.2. Construa uma árvore de decisão $h_m(x)$ com o novo *dataset* $\{(x_i, r_i^{(m)})\}_{i=1}^{n}$

3.3. Encontre o tamanho do passo a ser dado na descida do gradiente descendente:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)) \tag{2.27}$$

3.4. Atualize o modelo para a etapa atual de m com:

$$F_m(x) = F_{m-1} + \gamma_m \nu h_m(x) \tag{2.28}$$

onde ν é a taxa de aprendizado, ou *learning rate* em inglês, definido antes do treinamento (HASTIE; TIBSHIRANI; FRIEDMAN, 2001).

4. **Saída:** o resultado final do algoritmo é o somatório de todos os modelos gerados e seus passos no gradiente descendente:

$$F_T(x) = F_0(x) + \sum_{t=1}^{T} \gamma_t h_t(x) \tag{2.29}$$

Por fim, é importante ressaltar que há diferentes implementações do *gradient boosting*, sendo que uma das mais utilizadas é conhecida como *XGBoost*. Essa formulação incrementa o algoritmo original, pois dá suporte a computação de gradientes de segunda ordem, o que provê melhores indícios sobre a direção da derivada calculada e permite alcançar de maneira mais eficiente o valor mínimo da função de custo. Além disso, ela oferece regularização, o que aprimora a generalização dos modelos criados, e possibilita a realização de treinamentos em paralelo ou de forma distribuída, o que acelera o processo como um todo. (XGBOOST, 2021a)

2.3.2 Séries temporais

O termo séries temporais refere-se a todo conjunto de dados no qual exista uma "correlação introduzida pela amostragem de pontos adjacentes no tempo"(SHUMWAY; STOFFER, 2017). De forma mais geral, levando em conta a definição de HYNDMAN; ATHANASOPOULOS, uma série temporal trata-se de "uma lista de números, junto com algumas informações sobre o dia e horário em que esses números foram registrados".

A sociedade, de modo geral, usufrui dos benefícios de análises de séries temporais constantemente. De acordo com SHUMWAY; STOFFER, é possível listar inúmeras aplicações científicas, tais como:

1. Um epidemiologista interessado no número de casos de influenza observados durante um período de tempo;

2. Na medicina, as medidas de pressão arterial rastreadas ao longo do tempo podem ser úteis para determinar os medicamentos usados no tratamento da hipertensão;

3. Estudos de ressonância magnética podem ajudar cientistas a entender melhor os impulsos nervosos do cérebro humano ao longo do tempo quando estimulado sob circunstâncias experimentais.

2.3.2.1 Componentes de séries temporais

Um conjunto de dados de uma série temporal possui a característica de ser divisível em componentes que, quando combinados, restauram o dado originário. Esses componentes podem ser definidos como (HYNDMAN; ATHANASOPOULOS, 2018):

1. Tendência: entende-se como um aumento ou diminuição de longo prazo nos dados. Podem ocorrer mudanças de direção quando a tendência se altera de crescente para decrescente, por exemplo;

2. Sazonalidade: um padrão sazonal ocorre quando uma série temporal é afetada por fatores como a época do ano ou dias da semana. A sazonalidade é sempre de frequência fixa e conhecida;

3. Ciclos: um ciclo ocorre quando os dados exibem subidas e quedas que não possuem uma frequência fixa. Essas variações são geralmente devido às condições econômicas e muitas vezes relacionadas ao "ciclo de negócios";

4. Restante: a componente restante, conhecida em inglês como *remainder*, contém a parte restante que não se encaixa em nenhum dos três componentes descritos anteriormente.

Durante o processo de divisão de uma série temporal entre as componentes descritas, é comum combinar a tendência e o ciclo em apenas uma componente geralmente chamada tendência-ciclo, ou apenas tendência. Dessa forma, como mostra HYNDMAN; ATHANASOPOULOS, é possível decompor uma série temporal de duas maneiras: aditiva e multiplicativa.

A decomposição aditiva permite que uma série temporal seja escrita da seguinte maneira:

$$y_t = S_t + T_t + R_t \tag{2.30}$$

onde y_t são os dados, S_t é a componente sazonal, T_t é a componente tendência-ciclo e R_t é a componente restante para todo o período t. Em contrapartida, a decomposição multiplicativa permite que a série temporal seja escrita de acordo com a seguinte expressão matemática

$$y_t = S_t \cdot T_t \cdot R_t \tag{2.31}$$

A decomposição aditiva é a mais apropriada para os casos no qual a magnitude das variações em torno da tendência não oscilar de acordo com a média dos valores da série temporal. Quando a variação em torno da tendência parece ser proporcional a média dos valores da série temporal, então uma decomposição multiplicativa é mais apropriada. As decomposições multiplicativas são comuns com séries temporais de caráter econômico (HYNDMAN; ATHANASOPOULOS, 2018).

2.3.2.2 Previsão de séries temporais

Previsões (ou *forecasting*) de séries temporais são ferramentas importantes no planejamento e execução eficiente de decisões tomadas por parte de empresas, governos ou até mesmo indivíduos. Prever com precisão o consumo de energia elétrica em uma cidade tendo em vista seu crescimento populacional e desempenho econômico ou decidir a quantia de produtos necessários no estoque de uma empresa para a próxima festividade importante do ano são alguns exemplos de aplicação de previsão em séries temporais.

Dessa forma, a previsão de séries temporais pode ser dividida em duas categorias: qualitativa e quantitativa, definidas da seguinte maneira de acordo com HYNDMAN; ATHANASOPOULOS:

- Qualitativa: realizada nos casos em que não há dados disponíveis ou se os dados disponíveis não forem relevantes para a previsão;
- Quantitativa: pode ser aplicada se as seguintes condições forem satisfeitas:
    1. Informações numéricas sobre o passado estão disponíveis;
    2. É razoável supor que alguns aspectos dos padrões do passado continuarão no futuro.

Entre os modelos matemáticos de aprendizado de máquina comumente utilizados na previsão de séries temporais e referenciados pela literatura, é válido citar dois: modelos ARIMA e Redes Neurais Recorrentes ou *Recurrent Neural Networks* (RNNs), em inglês. As Sessões 2.3.3 e 2.3.4.1 abordarão com mais detalhes o funcionamento desses dois algoritmos.

2.3.2.3 Tratamento de *outliers* em séries temporais

O dado que compõe uma série temporal pode apresentar, em certos casos, valores faltantes ou com uma ordem de grandeza bastante distinta do que se observa como o padrão esperado. Essas ocorrências de pontos fora da curva, conhecidos popularmente como *outilers*, geralmente necessitam de algum tipo de tratamento para que não afete o treinamento de modelo preditivos com o dado, por exemplo.

Uma técnica robusta e eficiente usada para realizar esse procedimento consiste no uso de um filtro de Hampel (LIU; SHAH; JIANG, 2004). Esse método funciona percorrendo o dado (que no caso de uma série temporal, é uma sequência de números) com uma janela de tamanho K e um limiar n (MATHWORKS, 2021). De dentro dessa janela, composta de $x_{i-\frac{K}{2}}, ..., x_i, ..., x_{i+\frac{K}{2}}$, se calcula a mediana m_i dos valores e em seguida o desvio absoluto das medianas, definido por $dam_i = mediana(|x_{i-\frac{K}{2}} - m_i|, ..., |x_{i+\frac{K}{2}} - m_i|)$. Com esses valores, computa-se o desvio padrão $\sigma_i = S \cdot n$, onde S é dado por

$$S = \frac{1}{\sqrt{2} \cdot \text{erfc}^{-1}(\frac{1}{2})} = 1.4826 \tag{2.32}$$

e o filtro tem a saída dada por

$$x_i = \begin{cases} m_i, & \text{se } |x_i - m_i| > n \cdot \sigma_i \\ x_i, & \text{caso contrário} \end{cases} \tag{2.33}$$

2.3.3 ARIMA

Entre as abordagens possíveis para realizar previsão de séries temporais, uma das mais usadas se baseia no uso de modelos ARIMA (Autoregressive Integrated Moving Average) (HYNDMAN; ATHANASOPOULOS, 2018), que apesar de usualmente serem classificados como métodos de *machine learning*, tem origem de certa forma distinta, em fundamentos puramente estatísticos. Essa metodologia, que possui exemplos de aplicação em diversos contextos, como na previsão do preço da eletricidade (Contreras et al., 2003), do preço de ações (Ariyo; Adewumi; Ayo, 2014) e até da velocidade do vento (KAVASSERI; SEETHARAMAN, 2009), tem como objetivo descrever as autocorrelações no dado por meio da combinação de três componentes: autoregressão, diferenciação e média móvel.

2.3.3.1 Autoregressão

A autoregressão representa uma combinação linear de valores anteriores da variável de interesse, ou seja, é formada, como o próprio nome diz, por uma regressão de uma variável contra si mesma. Ela pode ser descrita pela seguinte fórmula (HYNDMAN; ATHANASOPOULOS, 2018):

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t \tag{2.34}$$

ε_t representa ruído branco ("white noise", em inglês), que são séries temporais sem autocorrelação, ou, de maneira mais formal, "uma sequência de variáveis aleatórias independentes e distribuídas de maneira idêntica com média e variância finitas"(TSAY, 2010).

p representa a ordem da autoregressão, ou seja, o número de termos anteriores de y a serem considerados.

2.3.3.2 Diferenciação

A diferenciação, que representa o segundo componente do modelo ARIMA, tem o objetivo de transformar uma série temporal não estacionária em estacionária.

Séries temporais não estacionárias são aquelas que possuem propriedades atreladas à sazonalidade ou a uma tendência (HYNDMAN; ATHANASOPOULOS, 2018). No entanto, em geral, é necessário que o dado seja estacionário para se realizar o cálculo dos modelos de autoregressão e média móvel, explicados respectivamente nas seções 2.3.3.1 e 2.3.3.3.

Nesses casos, uma maneira de remover as tendências que tornam um dado não estacionário é por meio da aplicação de diferenciação, que é a subtração dos valores de observações consecutivas.

Como nos outros dois componentes, essa operação também possui um grau, aqui representado pela variável d. No caso simples, onde $d = 1$, temos:

$$y'_t = y_t - y_{t-1} \tag{2.35}$$

Também é possível realizar a diferenciação uma segunda vez ($d = 2$), onde se teria

$$y''_t = y'_t - y'_{t-1} = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2} \tag{2.36}$$

É importante ressaltar que quase nunca é necessário ir além de diferenciações de segundo grau, como explicado por HYNDMAN; ATHANASOPOULOS.

2.3.3.3 Média móvel

O modelo de média móvel é similar ao modelo de autoregressão, porém, em vez de utilizar os valores anteriores da variável de previsão, usam-se os valores dos erros das q previsões anteriores, da seguinte forma:

$$y_t = c + \theta_1 \varepsilon_{t-1} + \theta 2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q} \tag{2.37}$$

Dessa maneira, assim como para a autroregressão, há um valor para a ordem (q) da média móvel representando o número de termos passados a se considerar no cálculo.

2.3.3.4 Modelo ARIMA não sazonal

Por meio da combinação dos três modelos apresentados acima, é possível a criação de um modelo ARIMA não sazonal, ou seja, que é capaz de realizar previsões a partir de dados que não apresentem algum tipo de periodicidade. Esse modelo, chamado de $ARIMA(p, d, q)$, pode ser escrito assim:

$$y'_t = c + \phi y'_{t-1} + ... + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + ... + \theta_q \varepsilon_{t-q} + \varepsilon_t \tag{2.38}$$

Onde y'_t é a série diferenciada para $d = 1$ e p e q são as ordens da autoregressão e da média móvel, respectivamente. No caso de algum desses valores ser 0, obtém-se casos especiais de modelos ARIMA, como um $ARIMA(0, 0, 0)$, que é ruído branco, um modelo de autoregressão ($ARIMA(p, 0, 0)$ ou $AR(p)$) ou um modelo de média móvel ($ARIMA(0, 0, q)$ ou $MA(q)$) (HYNDMAN; ATHANASOPOULOS, 2018).

2.3.3.5 Modelo ARIMA sazonal

Modelos ARIMA também são capazes de lidar com séries temporais sazonais. No entanto, para isso, é necessária a adição de quatro termos ao modelo ARIMA não sazonal mostrado na seção anterior.

Os três primeiros componentes novos, com ordem dada por P, D, Q, possuem sentido similar aos termos representados por p, d, q. Porém, têm o objetivo de capturar padrões de ocorrência periódica. Já o quarto termo (m) descreve o número de observações por período.

Dessa maneira, para facilitar a definição do modelo sazonal, é possível usar a seguinte notação com um operador B, indicando a ação de retroceder determinado y_t em n períodos:

$$B^n y_t = y_{t-n} \tag{2.39}$$

Então, por exemplo, a aplicação de dois desses operadores em sequência, como abaixo:

$$B(By_t) = B^2 y_t = y_{t-2} \tag{2.40}$$

Significaria retroceder y_t dois períodos.

Sendo assim, é possível escrever um modelo $ARIMA(p, d, q)(P, D, Q)m$ da seguinte forma (HYNDMAN; ATHANASOPOULOS, 2018):

$$(1 - \phi_1 B - ... - \phi_p B^p)(1 - \Phi_1 B^m ... - \Phi_P B^{mP})(1 - B)^d (1 - B^m)^D y_t = (1 + \theta_1 B + ... + \theta_q B^q)(1 + \Theta_1 B^m + ... + \Theta_Q B^{mQ})\varepsilon_t \tag{2.41}$$

2.3.3.6 Treinamento de modelos ARIMA

O treinamento de um modelo ARIMA (sazonal ou não) consiste na estimativa dos coeficientes não conhecidos que compõem as equações apresentadas nas seções anteriores.

No entanto, a escolha do método para descoberta desses valores depende, entre outros pontos, dos parâmetros p, d, q, P, D, Q e m escolhidos inicialmente na configuração do modelo. Nesse sentido, é possível utilizar maneiras mais simples e eficientes computacionalmente dependendo dos componentes escolhidos para o modelo.

No caso de um $ARMA(p, d)$, por exemplo, uma abordagem possível é a utilização de estimativa por máxima verossimilhança (BARTLETT, 2010). De maneira simples, essa técnica tem o objetivo de encontrar valores para os coeficientes do modelo de modo a maximizar a probabilidade de obter uma distribuição com o dado observado, ou seja, tomando um conjunto de amostras $Y = (y_1, ..., y_t)$ de tamanho t, o vetor de parâmetros $\beta = [\phi_1, ..., \phi_p, \theta_1, ..., \theta q, \sigma^2]^T$ e considerando a função de probabilidade conjunta:

$$f(y_t, y_{t-1}, ..., y_1; \beta) \tag{2.42}$$

Temos uma função de verossimilhança dos parâmetros β dado Y na forma:

$$L(\beta|Y) = f(y_t, y_{t-1}, ..., y_1; \beta) \tag{2.43}$$

Dessa forma, com base no *log* da equação acima, o que facilita sua diferenciação, temos como objetivo maximizar:

$$log(L(\beta|Y)) = l(\beta|Y) \tag{2.44}$$

Para isso, assumindo que a derivada de $l(\beta|Y)$ existe e é contínua para qualquer β, é necessário contemplar a condição:

$$\frac{\partial l(\beta|Y)}{\partial \beta} = 0 \tag{2.45}$$

No contexto de um processo $ARMA$, por exemplo, assumindo-se que os erros e_t se tratam de ruído branco (*white noise*), ou seja, tratam-se de variáveis geradas por uma distribuição normal com média 0 e variância σ^2, tem-se que a verossimilhança gaussiana exata é:

$$L(\beta|Y) = (2\pi)^{-\frac{t}{2}}|\Gamma(\beta)|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}Y^T\Gamma(\beta)^{-1}Y\right\} \tag{2.46}$$

onde $\Gamma(\beta)$ é a matriz de covariância $t \times t$ de Y dependente de β. Aplicando *log* dos dois lados, obtém-se a função que se deseja maximizar para obter os valores de β:

$$l(\beta|Y) = \log(L(\beta|Y)) = -\frac{1}{2}\left[t\log(2\pi) + \log(|\Gamma(\beta)|) + Y^T\Gamma(\beta)^-1Y\right] \tag{2.47}$$

2.3.3.7 Escolha dos parâmetros do ARIMA

A escolha dos parâmetros de um modelo ARIMA (p, d, q) geralmente se baseia na seguinte abordagem (HYNDMAN; ATHANASOPOULOS, 2018):

1. Avaliar se a série temporal é estacionária ou não. Se sim, é necessário aplicar diferenciação, o que determina o valor de d;
2. Avaliar os gráficos ACF (*Autocorrelation Function*) e PACF (*Partial Autocorrelation Function*) para determinar o valor de p e q. Um exemplo prático desses gráficos pode ser encontrado na Seção 4.4.1.1, nas Figuras 17 e 18.

Para realizar o **primeiro passo**, há algumas estratégias que podem ser usadas em conjunto. Primeiramente, uma avaliação visual da série temporal pode identificar facilmente se o dado apresenta tendências ou sazonalidade. Outra técnica usada é avaliar o gráfico ACF (HYNDMAN; ATHANASOPOULOS, 2018). Esse gráfico, baseado na autocorrelação, permite calcular a relação linear entre os valores defasados de uma série temporal, na forma:

$$r_k = \frac{\sum_{t=k+1}^{T}(y_t - \overline{y})(y_{t-k} - \overline{y})}{\sum_{t=1}^{T}(y_t - \overline{y})} \tag{2.48}$$

Como explicado por HYNDMAN; ATHANASOPOULOS, no caso do dado possuir tendências, "os valores de autocorrelação para pequenos atrasos (*lags*) tendem a ser grandes e positivos porque observações próximas também tem um valor similar". Dessa maneira, o gráfico ACF nesses casos decrementa vagarosamente.

No contexto de dados sazonais, é observável um aumento dos valores de autocorrelação nos intervalos onde há o início/fim das ocorrências de periodicidade.

Além disso, é possível realizar testes estatísticos para determinar a necessidade de diferenciação no dado. Um dos testes que pode ser usado é o de Ljung-Box (HYNDMAN; ATHANASOPOULOS, 2018), representado por:

$$Q^* = T(T+2)\sum_{k=1}^{h}(T-k)^{-1}r_k^2 \tag{2.49}$$

Onde T é o número de observações e h o número do atraso máximo considerado.

Valores de Q^* grandes indicam que as autocorrelações não são produzidas por uma série de ruído branco e, consequentemente, os resíduos ou erros cometidos pelo modelo ainda possuem informações não aleatórias que podem ser usadas para realizar as predições.

Já a **segunda etapa** consiste na avaliação de dois gráficos: ACF e PACF. Para isso, assume-se que o dado é originado de um modelo $ARIMA(p,d,0)$ ou $ARIMA(0,d,q)$, ou seja, que não se trata de uma situação onde p e q são positivos.

O valores de autocorrelação parcial podem ser estimados com o último coeficiente do modelo de autoregressão, ou seja, α_k, o k-ésimo valor de autocorrelação parcial, é igual a estimativa de ϕ_k em um modelo $ARIMA(k,0,0)$.

Dessa forma, em um modelo $ARIMA(p,d,0)$, os padrões observados nos gráficos são:

- ACF decai exponencialmente (de maneira lenta) ou apresenta um padrão senoidal;
- Há um pico significante no atraso p no gráfico PACF, mas nenhum após isso.

Já um modelo $ARIMA(0,d,q)$ apresentaria os seguintes padrões:

- PACF decai exponencialmente (de maneira lenta) ou apresenta um padrão senoidal;
- Há um pico significante no atraso p no gráfico ACF, mas nenhum após isso.

Para o caso de modelos ARIMA sazonais, o procedimento é o mesmo, apenas considerando o pico e o decaimento exponencial nos atrasos sazonais. Então, em um modelo $ARIMA(0,0,0)(0,0,1)_{12}$, por exemplo, haveria um pico no atraso *12* do gráfico ACF e nenhum após, assim como uma queda gradual dos valores do PACF a cada conjunto de 12 valores.

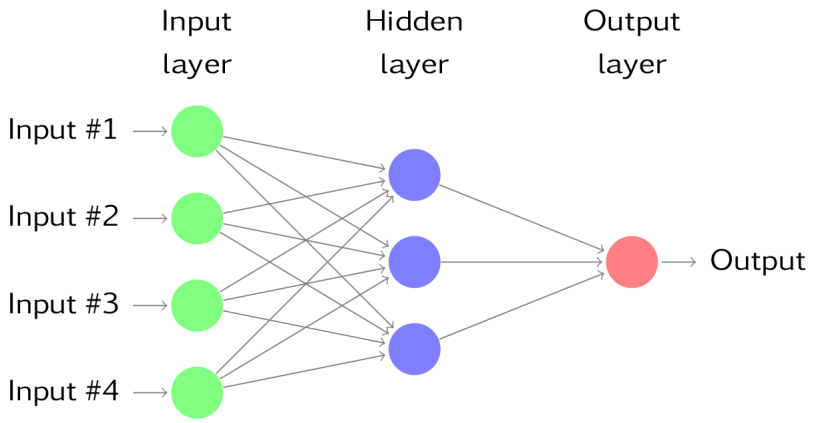
2.3.4 Redes Neurais Artificiais

Inspiradas de maneira extremamente vaga no funcionamento de um cérebro real, as Redes Neurais Artificiais (RNAs) são aproximadores universais de funções matemáticas, formados por "redes de elementos de processamento simples operando em seus dados locais e se comunicando com outros elementos"(SVOZIL; KVASNICKA; POSPICHAL, 1997).

Ainda de maneira mais específica, as RNAs podem ser entendidas como uma rede de operadores matemáticos organizados nas seguintes camadas (HYNDMAN; ATHANASOPOULOS, 2018):

- Camada de entradas (*input layer*): recebe como entrada o conjunto de variáveis de previsão ou independentes;
- Camada escondida (*hidden layer*): recebe como entrada a saída de camadas anteriores, podendo essas serem outras camadas escondidas ou a camada de entrada;
- Camada de saída (*output layer*): recebe como entrada a saída da última camada escondida e sua saída representa a predição para a variável dependente (ou de interesse) do problema.

Figura 8 – Ilustração da arquitetura de RNAs



Fonte: (HYNDMAN; ATHANASOPOULOS, 2018)

A Figura 8 exibe uma arquitetura genérica de uma rede neural multicamada, que se baseia em um fluxo de alimentação unidirecional (em inglês *multilayer feed-forward neural network* ou MFFNN), onde cada nó da camada j recebe como entrada a saída da camada anterior $j-1$. Dessa forma, as entradas de cada camada escondida são linearmente combinadas com os pesos das conexões entre os nós. Por fim, o resultado pode ser alterado por uma função não linear antes de ser movido para a camada seguinte.

Nesse contexto, pode-se estabelecer que as entradas de cada nó da *hidden layer* seriam combinados da seguinte maneira (HYNDMAN; ATHANASOPOULOS, 2018):

$$z_j = b_j + \sum_{i=1}^{4} w_{i,j} x_i \tag{2.50}$$

E o valor contido em cada z_j poderia ser modificado por uma função não linear, como a *sigmoid*, por exemplo:

$$s(z_j) = \frac{1}{1 + e^{-z_j}} \tag{2.51}$$

Por fim, tem-se que o objetivo do treinamento de uma rede neural é ajustar seus parâmetros, no caso apresentado acima, $b1$, $b2$, $b3$ e $w_{1,1}$, ..., $w_{4,3}$, a partir do conjunto de dados de treino, a fim de minimizar o erro calculado por alguma função de custo escolhida previamente.

Vale notar que RNA's também são úteis para predições de séries temporais. Nesse caso, a melhor abordagem se baseia no particionamento da série em janelas de predições, onde cada janela é utilizada como sequência de entrada da rede, tomando como objetivo a previsão de um ou mais pontos subsequentes da janela. Entretanto, como mostra (SVOZIL; KVASNICKA; POSPICHAL, 1997), redes neurais com essa abordagem tradicional de alimentação unidirecional dos dados "ignoram a ordem temporal dentro das janelas de entrada e cada nova entrada é considerada isoladamente. Nenhum estado é transportado das entradas anteriores para as etapas de tempo futuras".

Portanto, visando modelar problemas com dados baseados na sequência de valores (o que inclui situações em que há a presença de uma dimensão temporal), uma classe especial de algoritmos foi desenvolvida. Chamadas de Redes Neurais Recorrentes (em inglês *Recurrent Neural Networks*), essas RNA's, por meio do armazenamento e da retroalimentação de estados previamente aprendidos, são capazes de utilizar eventos ou valores passados na predição de sequências futuras.

2.3.4.1 *Recurrent Neural Networks*

Recurrent Neural Networks são tipos especializados de Redes Neurais Artificiais "caracterizadas por autoconexões internas, que podem, em princípio, modelar qualquer sistema dinâmico não linear, até um determinado grau de precisão"(BIANCHI et al., 2017).

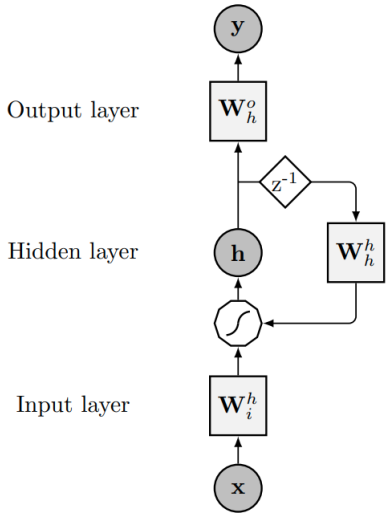
Tendo em vista sua capacidade de considerar a dimensão temporal em dados no qual a ordem dos eventos é relevante, as RNNs possuem muitas áreas de aplicação, como transdução de sequência (GRAVES, 2012), modelagem de áudio (OORD et al., 2016) e geração de imagens (GREGOR et al., 2015).

As seções subsequentes explicarão melhor a arquitetura, o processo de treinamento comumente utilizados e as equações matemáticas por trás dessa classe de redes neurais com atributos únicos.

2.3.4.1.1 Arquitetura de RNNs

A imagem a seguir ilustra a arquitetura de uma Rede Neural Recorrente simples.

Figura 9 – Ilustração da arquitetura de RNNs.



Fonte: (BIANCHI et al., 2017)

Conforme mostra a Figura 9, as RNNs podem ser entendidas como um grafo cíclico com nós representando as camadas de entrada, escondida e de saída semelhante as RNAs descritas anteriormente. Os retângulos com as identificações W_i^h, W_h^h e W_h^o são os pesos das camadas de entrada, escondida e saída respectivamente. Tais pesos são aprendidos durante o processo de treinamento. O polígono decágono representa a função não-linear aplicada pelos nós (sendo comumente utilizada a tangente hiperbólica [1]) e o losango z^{-1} a unidade do atraso temporal (BIANCHI et al., 2017).

2.3.4.1.2 Treinamento de RNNs

O treinamento de uma RNN depende do particionamento da série temporal a ser modelada em janelas de predição. Essas janelas dependem da escolha arbitrária de três atributos: largura de entrada (*input width*), largura do rótulo (*label width*) e comprimento do deslocamento (*offset*) (TENSORFLOW, 2021b).

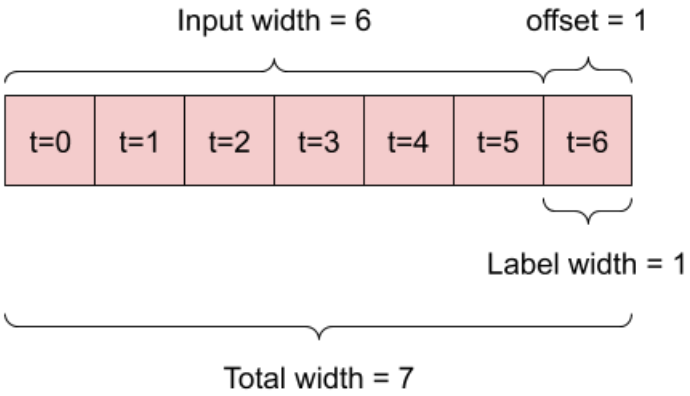
A imagem a seguir ilustra um particionamento hipotético de uma série temporal com o objetivo de realizar previsões 1h no futuro, dado 6 horas de histórico.

No exemplo ilustrado na Figura 10, a série temporal seria dividida em múltiplas janelas com os seguintes atributos e seus respectivos valores:

- largura de entrada: 6 unidades;
- largura do rótulo: 1 unidade;
- comprimento do deslocamento: 1 unidade,

[1] $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Figura 10 – Exemplo de particionamento de série temporal em janelas



Fonte: (TENSORFLOW, 2021b)

onde cada unidade representa uma única observação do dados disponível parar o treinamento e os retângulos $t = 0$ ... $t = 6$ caracterizam a ordem temporal dessas observações.

O treinamento de uma rede neural (tanto para RNAs quanto para RNNs) "consiste em modificar seus parâmetros por meio de uma otimização utilizando gradiente descendente, que minimiza uma dada função de perda que quantifica a precisão da rede em realizar a tarefa desejada". (BIANCHI et al., 2017)

Dessa maneira, o ajuste dos parâmetros citados na Figura 9 atinge a convergência através da repetição de dois passos, que ocorrem em ciclos conhecidos como **épocas**:

1. o cálculo de uma função custo L_k configurado com os pesos W_k;
2. a propagação dos novos pesos atingidos através do gradiente descendente $\frac{\partial L_k}{\partial W_k}$ após um conjunto X_k de entradas (chamado de *batch*) tiver sido processado (método chamado retropropagação através do tempo, ou em inglês *Backpropagation Through Time*).

A função custo L_k pode ser definida como

$$L_k = E\left(X_k, Y_k^*; W_k\right) + R_\lambda \tag{2.52}$$

onde E é a função que calcula o erro entre a previsão e a resposta desejada Y_k^* para uma entrada X_k utilizando os parâmetros W_k e R_λ uma função de regularização (BIANCHI et al., 2017).

Para o caso de regressões, por exemplo, uma função do cálculo do erro E comumente utilizada é a Média do Quadrado dos Erros (*Mean Squared Error* ou MSE) definida como:

$$MSE\left(Y_k, Y_k^*\right) = \frac{1}{\mid X_k \mid} \sum_{x \in x_k} \left(y_x - y_x^*\right)^2 \tag{2.53}$$

onde $y_k \in Y_k$ é a previsão realizada pela RNN para uma entrada $x_k \in X_k$ e $y_k^* \in Y_k^*$ a resposta no qual deseja-se que a rede aprenda.

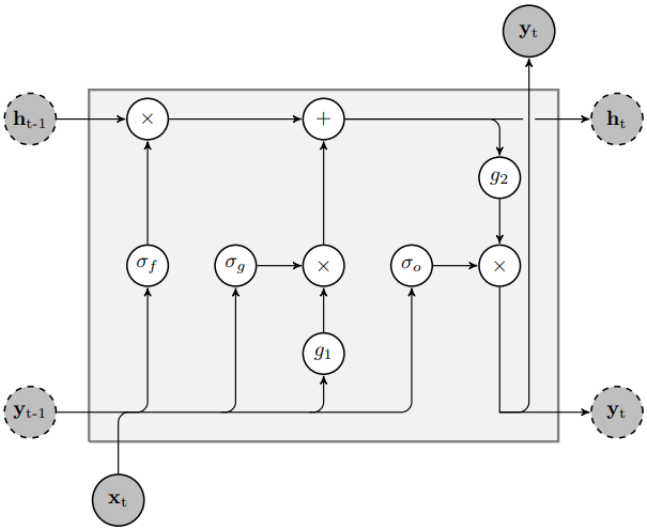
2.3.4.2 *Long Short-Term Memory*

O modelo arquitetural *Long Short-Term Memory* (LSTM) é um tipo de RNN caracterizado pela capacidade de modelar dados com dependência de curto a longo prazo. Nas palavras dos idealizadores dessa arquitetura de RNN (HOCHREITER; SCHMIDHUBER, 1997) "com a retropropagação através do tempo de RNNs convencionais, os sinais de erro que fluem para trás no tempo tendem a (1) explodir ou (2) desaparecer: a evolução temporal do erro de retropropagação depende exponencialmente do tamanho dos pesos".

Dessa maneira, o caso (1) pode levar a uma oscilação dos pesos dificultando a generalização da rede neural. Já no caso (2) aprender as correlações em dados de longo prazo pode levar uma quantia de tempo proibitiva ou não funcionar. (HOCHREITER; SCHMIDHUBER, 1997)

A Figura 11 ilustra o funcionamento de uma célula de uma rede neural recorrente com modelo arquitetural LSTM.

Figura 11 – Ilustração da arquitetura de uma célula no modelo LSTM



Fonte: (BIANCHI et al., 2017)

As variáveis h_{t-1}, h_t, y_{t-1} e y_t dentro dos círculos cinza escuros com linhas tracejadas representam o estado interno da célula. As funções σ_f, σ_g e σ_o representam as funções não-lineares sigmóides (Equação 2.51) usadas nas etapas de esquecimento, atualização e portas de saída respectivamente. Os operadores ponto a ponto $+$ e x representam a parte

linear da célula. Por fim, g_1 e g_2 são funções não-lineares geralmente implementadas como tangente hiperbólica.

3 Metodologia

Com o objetivo de desenvolver um sistema capaz de realizar a estimativa de curvas de carga e predições do consumo de energia utilizando os dados gerados e disponibilizados pela plataforma SIGE, este trabalho se caracteriza como uma pesquisa experimental aplicada. Dessa maneira, através da definição do problema e do objeto de estudo, foi realizado o levantamento do referencial bibliográfico em especial para os tópicos de *machine learning* aplicado em regressão e séries temporais.

Ao final da pesquisa de modelos que se encaixem adequadamente ao contexto, pretende-se construir o módulo apto em gerar perfis de carga de consumo elétrico. Esse sistema permitirá predições capazes de se adaptar aos diferentes padrões contidos no dado, consumidos de maneira automática e em tempo real da plataforma SIGE, ao longo do tempo.

3.1 Levantamento Bibliográfico

As referências bibliográficas deste trabalho são baseadas em artigos, livros, monografias e outros tipos de publicações comumente aceitos pela comunidade científica acerca dos métodos de *machine learning* mais utilizados em regressões e séries temporais com informações quantitativas sobre a demanda de carga em um sistema. Além disso, foram pesquisados materiais que explicassem a arquitetura e o funcionamento de tais métodos bem como seus passos de treinamento e avaliação.

Por fim, foram definidos as etapas para realizar experimentos acerca do objeto de estudo com o propósito de obter resultados iniciais que ajudarão a decidir o(s) método(s) de *machine learning* mais adequado(s) para a resolução da problemática.

3.2 Etapas dos experimentos

A criação de modelos capazes de realizar a determinação de curvas de carga se baseia nas seguintes etapas:

- **Etapa 1**: Coleta e análise exploratória dos dados registrados pelos sensores;
- **Etapa 2**: Tratamento de *outliers*;
- **Etapa 3**: Separação do dado de treino e teste;
- **Etapa 4**: Escolha e treinamento dos modelos;

- **Etapa 5**: Avaliação dos modelos.

3.2.1 Etapa 1: Coleta e análise de dados

Na primeira etapa, o objetivo é coletar os dados de consumo de energia medidos pelos sensores espalhados pelos campi da UnB.

Após isso, é necessária a realização de uma análise exploratória, buscando avaliar algumas hipóteses:

1. O dado pode ser tratado como uma série temporal?
2. O dado apresenta tendências sazonais? Se sim, qual a frequência dessas tendências?
3. Há diferenças significativas dos valores e das tendências de edifícios diferentes?
4. O dado apresenta "pontos fora da curva"(*outliers*)?

3.2.2 Etapa 2: Tratamento de *outliers*

Com a análise realizada na seção 3.2.1, é avaliada a presença de *outliers* no conjunto de dados. Caso haja a ocorrência desses pontos fora da curva, é usado um algoritmo para o seu tratamento.

3.2.3 Etapa 3: Separação do dado de treino e teste

O dado obtido na etapa anterior é separado em dois conjuntos: um de treinamento e outro para realizar predições e calcular o valor da métrica de avaliação escolhida.

3.2.4 Etapa 4: Escolha e treinamento dos modelos

Tomando como base a literatura e as conclusões obtidas na seção 3.2.1, é feita a escolha e o treinamento de modelos apropriados para a identificação de curvas de carga.

3.2.5 Etapa 5: Avaliação dos modelos

Na quarta etapa, após o treinamento, uma métrica é usada para avaliar a qualidade das predições feitas pelos modelos.

3.3 Etapas de desenvolvimento

A construção do SIGEML, que irá se comunicar com a API do SIGE e prover ao usuário uma interface que possibilite a geração de curvas de carga, retreinamento dos

modelos de *machine learning* e realização de previsões acerca do consumo de energia, se baseia nas seguintes etapas:

- **Etapa 1**: Levantamento de requisitos;
- **Etapa 2**: Definição da arquitetura de software;
- **Etapa 3**: Escolha do processo de desenvolvimento;
- **Etapa 4**: Definição e organização do *backlog* de tarefas;
- **Etapa 5**: Implementação e testagem;
- **Etapa 6**: *Deploy* do sistema.

3.3.1 Etapa 1: Levantamento de requisitos

Com o objetivo de entender melhor os detalhes de funcionamento do sistema, a primeira etapa é marcada pela definição e priorização de requisitos funcionais e não funcionais. Esse levantamento pode ser realizado escolhendo métodos tais como questionários, *brainstormings* e entrevistas com *stakeholders*.

3.3.2 Etapa 2: Definição da arquitetura de software

Após o esclarecimento e definição do escopo alcançados na Etapa 1 de desenvolvimento (Seção 3.3.1), inicia-se a definição de uma arquitetura de software que atenda aos requisitos. Para tanto, são construídos artefatos tais como diagramas e documentos que detalham a interação entre os possíveis sistemas a serem construídos e os já existentes. Ainda nessa etapa, são escolhidas as ferramentas que, integradas a arquitetura, permitirão o funcionamento do sistema atendendo tanto requisitos funcionais quanto não funcionais.

3.3.3 Etapa 3: Escolha do processo de desenvolvimento

Na terceira etapa, através de uma pesquisa sobre métodos de desenvolvimento de software, é escolhido uma metodologia que permita um desenvolvimento rápido, eficiente e com foco em qualidade tendo em vista as limitações de tempo e pessoal. Tal metodologia pode ser formada pela combinação de múltiplos métodos de desenvolvimento de software.

3.3.4 Etapa 4: Definição e organização do *backlog* de tarefas

Nessa etapa é criado o *backlog* do produto fragmentando o escopo que se deseja alcançar em tarefas com critérios de aceitação. Essas tarefas podem ser gerenciadas na organização criada no GitHub que contém o código produzido neste trabalho.

3.3.5 Etapa 5: Implementação e testagem

Com as tarefas definidas, inicia-se a implementação do sistema e a integração dos modelos escolhidos a ele, adotando uma abordagem de testagem iterativa. O critério de aceitação das tarefas se baseia na qualidade das funcionalidade desenvolvidas e entregues, sendo que um dos parâmetros para integrar a *feature* pode ser a existência de testes automatizados.

3.3.6 Etapa 6: *Deploy* do sistema

Por fim, o sistema produzido na etapa 5 de desenvolvimento (Seção 3.3.5) é colocado em produção realizando o *deploy* numa infraestrutura adequada. Entretanto, essa etapa pode ser realizada em paralelo com a anterior através de práticas conhecidas pela comunidade de software tais como Integração Contínua (em inglês *Continuous Integration*) e Entrega Contínua (em inglês *Continuous Deployment*).

3.4 Ferramentas

As ferramentas utilizadas para a realização dos experimentos (Seção 3.2) e desenvolvimento do sistema (Seção 3.3) são:

- **FastAPI**: "*FastAPI* é um *framework web* moderno, de alta perfomance para construir API's com *Python* 3.6+ baseando-se nos *type hints* padrões do *Python*"(RAMíREZ, 2021);
- **Jupyter Notebook**: pacote para a linguagem de programação *Python* de código aberto que permite criar e compartilhar documentos que contêm código ativo, equações, visualizações e texto narrativo. Os usos incluem: limpeza e transformação de dados, simulação numérica, modelagem estatística, visualização de dados, aprendizado de máquina e muito mais (JUPYTER, 2021);
- **Keras**: API construída sobre o *Tensorflow*, projetada para possuir uma carga de aprendizado e tempo de codificação menor ao se construir redes neurais (KERAS, 2021a);
- **Matplotlib**: biblioteca abrangente para a criação de visualizações estáticas, animadas e interativas em *Python* (MATPLOTLIB, 2021);
- **Mlflow**: "uma plataforma *open source* para gerenciar o ciclo de vida de *machine learning*, incluindo experimentação, reprodutibilidade, *deploy* e um *registry* central de modelos"(MLFLOW, 2021);
- **Mosquitto**: "um *broker* MQTT de código aberto"(ECLIPSE, 2021);

- **Pandas**: pandas é uma biblioteca de código aberto licenciada por BSD que fornece estruturas de dados de alto desempenho e fáceis de usar e ferramentas de análise de dados para a linguagem de programação *Python* (PANDAS, 2021);
- **PostgreSQL**: "o banco de dados relacional de código aberto mais avançado do mundo"(POSTGRESQL, 2021);
- **Python**: linguagem de programação interpretada, de alto nível, orientada a objetos e dinamicamente tipada (PYTHON, 2021);
- **React JS**: "uma biblioteca JavaScript para construir interfaces de usuário". (REACTJS, 2021);
- **Redis**: "Redis é uma estrutura de armazenamento de dados *in-memory* de código aberto, usado como banco de dados, cache e *broker* de mensagens". (REDIS, 2021);
- **Scikit-learn**: biblioteca em *Python* que disponibiliza "ferramentas simples e eficientes para análise de dados preditiva"(SCIKITLEARN, 2021b);
- **Statsmodels**: módulo *Python* que fornece classes e funções para a estimativa de muitos modelos estatísticos diferentes, bem como para a realização de testes estatísticos e exploração de dados estatísticos (STATSMODELS, 2021);
- **TensorFlow**: plataforma de código aberto de ponta a ponta para aprendizado de máquina. Ele tem um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos da comunidade que permite aos pesquisadores impulsionar o estado da arte em ML e aos desenvolvedores criar e implantar facilmente aplicativos baseados em ML (TENSORFLOW, 2021a);
- **xgboost**: "*XGBoost* é uma biblioteca de *gradient boosting* projetada para ser altamente eficiente, flexível e portátil"(XGBOOST, 2021b).

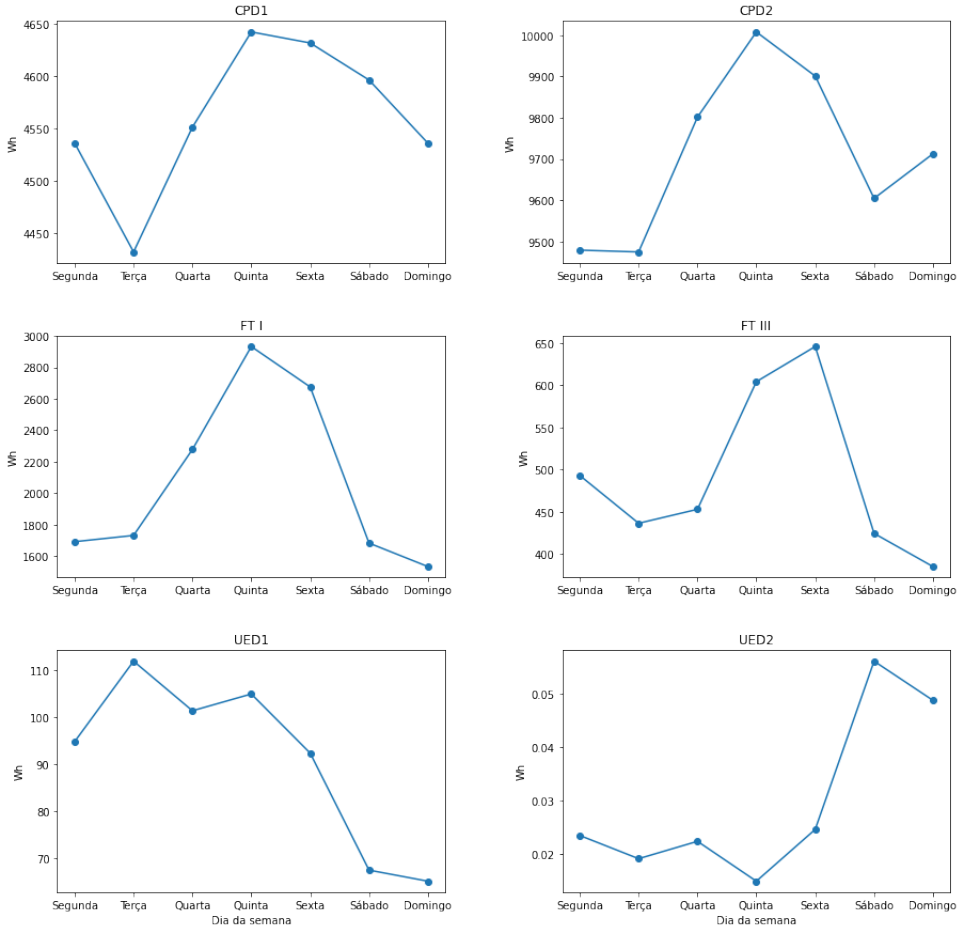
4 Experimentos Iniciais

Essa seção apresenta os resultados experimentais [1] obtidos tomando como base as etapas descritas na Seção 3.2.

4.1 Etapa 1: Coleta e análise de dados

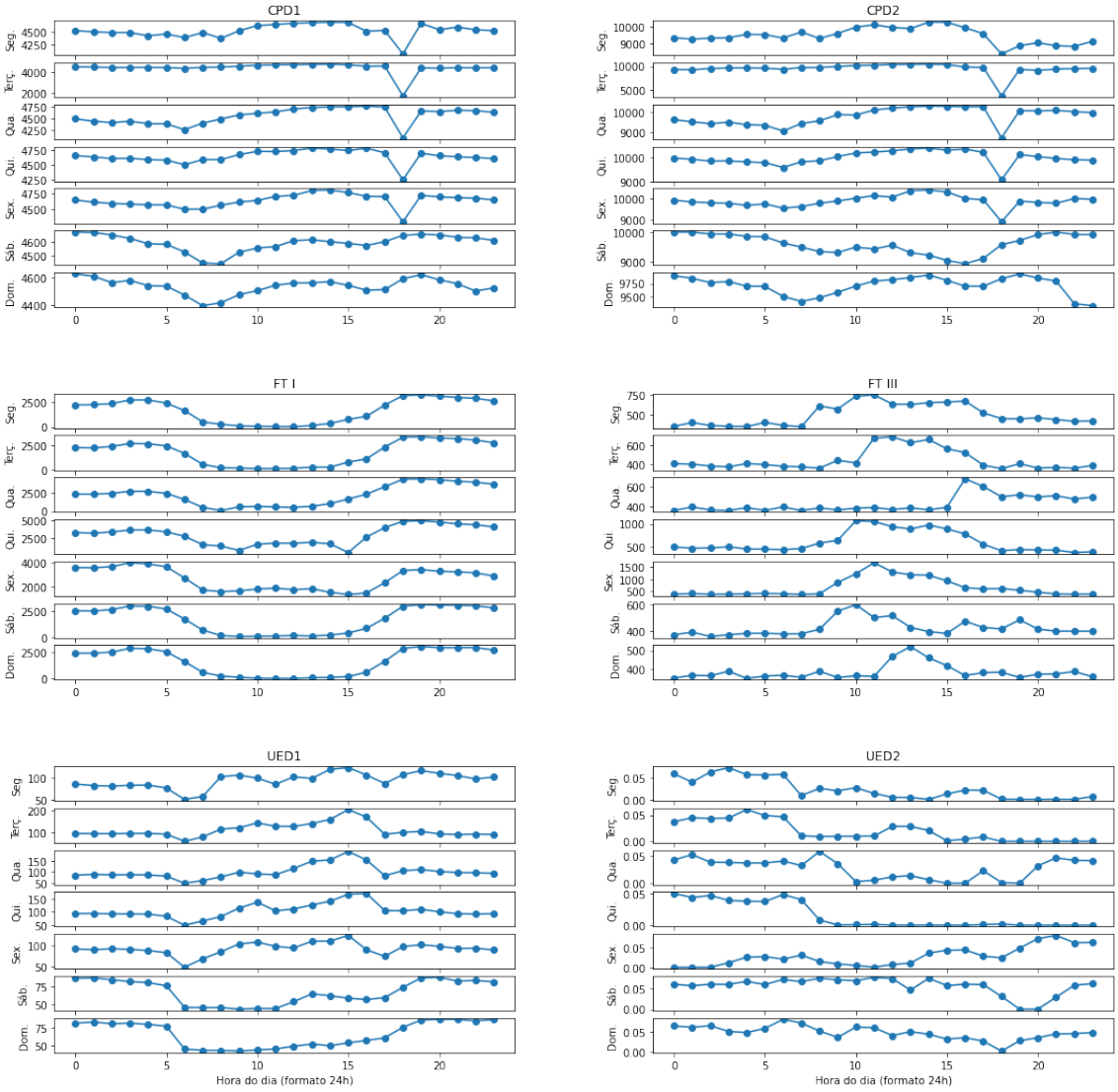
Através de um *script* escrito na linguagem de programação *Python*, disponível nos servidores da plataforma SIGE, foram coletados os dados de consumo, geração e potência de energia elétrica dos medidores e gerados gráficos a fim de entender os padrões de consumo energético de alguns prédios da UnB. As Figuras 12 e 13, por exemplo, mostram as médias de consumo de energia elétrica para cada dia da semana e por hora do dia respectivamente. Já na Figura 14, são exibidos *boxplots* para o consumo de cada dia da semana, com a indicação dos quartis e dos *outliers* contidos no dado.

Figura 12 – Consumo médio de energia em Wh por dia da semana para cada medidor



[1] Os notebooks com o código fonte dos experimentos podem ser encontrados aqui

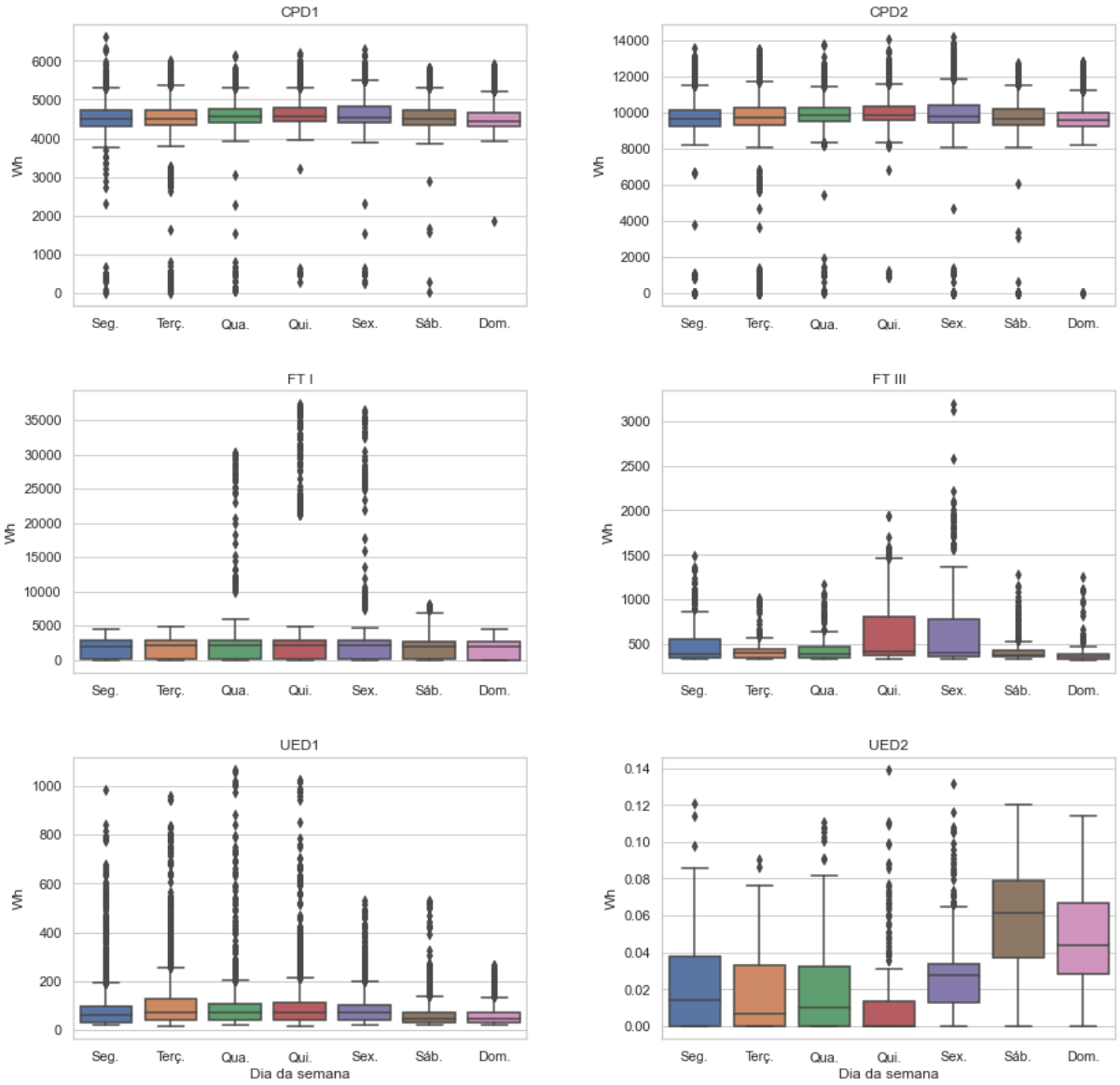
Figura 13 – Consumo médio de energia em Wh por hora do dia para cada medidor



Com base nas Figuras 12 e 13 é possível perceber que os dados seguem alguns padrões de sazonalidade tanto para os dias da semana quanto para as horas do dia. Além disso, o gráfico da Figura 13 mostra alguns exemplos de como esse padrão de sazonalidade pode ser diferente entre edifícios distintos da UnB. Por exemplo, devido ao período adverso de isolamento social imposto pela pandemia do vírus COVID-19, o medidor 1 da Faculdade de Tecnologia (FT1) apresenta um consumo mínimo em horário comercial e máximo durante o período da noite. Já o medidor 1 do Centro de Processamento de Dados da UnB (CPD1) apresenta um consumo quase que constante por se tratar de um prédio que contém múltiplos computadores servidores com aparelhos de ar-condicionado para controle da temperatura.

Além disso, analisando o gráfico apresentado na Figura 14, percebe-se que para maior parte dos dias, há a presença bem significativa de *outliers*, ou seja, medidas que estão fora dos limites mínimo e máximo representados por $Q1 - 1.5 * IQR$ e $Q3 + 1.5 * IQR$

Figura 14 – Distribuições dos consumos médios de energia em Wh por dia da semana e para cada medidor



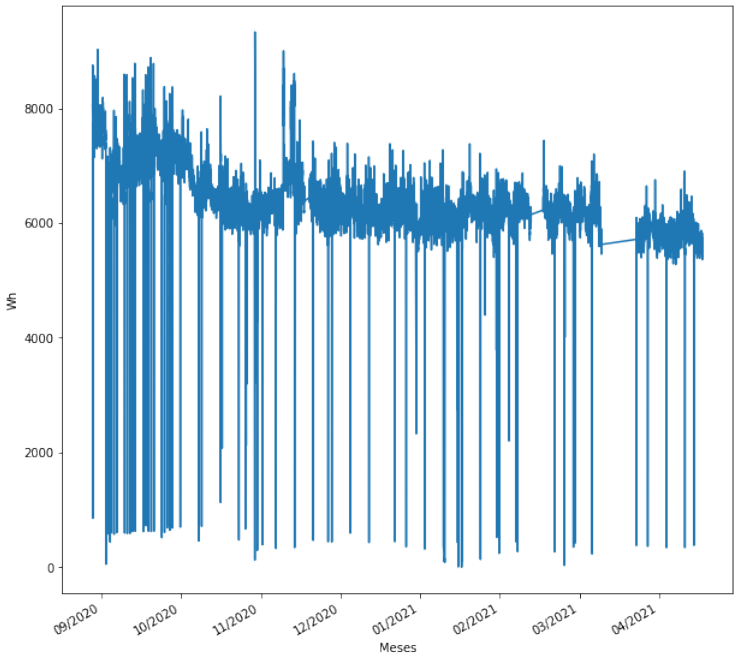
respectivamente, sendo IQR a distância interquartil. [2]

Ainda nessa etapa, foram selecionados os dados coletados pelo sensor 1 do CPD da UnB (CPD1), entre o período de 16 de setembro de 2020 até 04 de maio de 2021, para dar continuidade as etapas subsequentes. Essa decisão foi tomada considerando o medidor com o maior volume de dados e com características mais próximas do uso comum antes do período de pandemia do novo coronavírus COVID-19. A Figura 15 mostra todo o conjunto de dados disponível para análise do medidor CPD1:

De acordo com a Figura 15 percebe-se que, assim como explicado na Seção 2.3.2,

[2] Quartis são três valores que separam um conjunto de dados ordenado em partes iguais, cada parte representando 1/4 do dado. A distância interquartil (*interquartile range* ou IQR) representa a diferença entre o terceiro e primeiro quartil (NCERT, 2018).

Figura 15 – Consumo de energia elétrica - CPD1



que o conjunto de dados do medidor CPD1 escolhido para análise pode ser considerado uma série temporal. Além disso, observando o gráfico, reafirmou-se a presença de *outliers*, indicando a necessidade de algum tratamento adequado.

4.2 Etapa 2: Tratamento de *outliers*

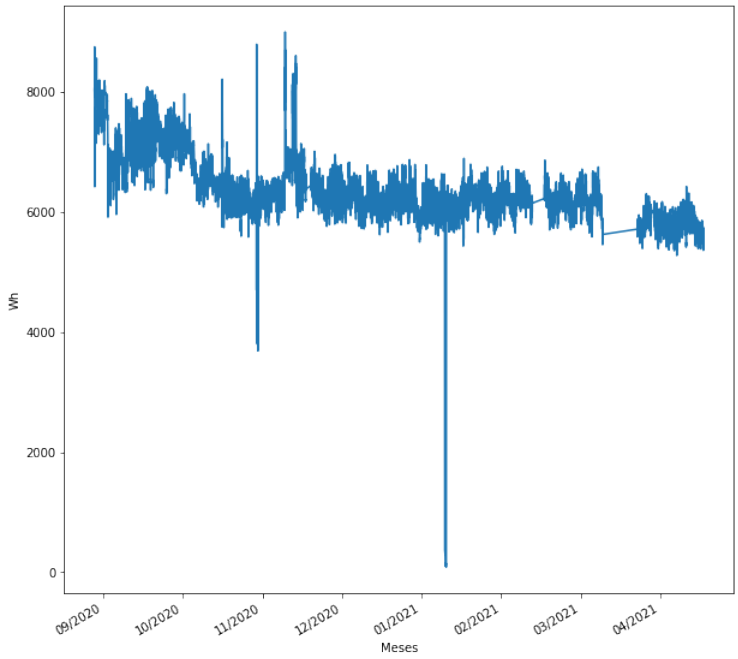
Como foi constada a presença de *outliers* no dado analisado anteriormente, foi necessário a utilização de um algoritmo capaz de filtrar e substituir valores que possuam uma variação muito grande em relação as medidas próximas a eles. Para isso, foi aplicado um filtro de Hampel (explicado em mais detalhes na Seção 2.3.2.3) com uma janela de tamanho 16 e um *threshold* $n = 2$. O dado após essa modificação se apresenta no formato mostrado na Figura 16.

4.3 Etapa 3: Separação do dado de treino e teste

Após a remoção dos *outliers*, foi feita a divisão do dado para ser utilizado no treinamento dos modelos e do dado para a avaliação.

Para os modelos de predição do consumo energético, a separação foi feita tomando como base um critério cronológico. Dessa forma, 70% do dado (13562 amostras), compreendendo as medições de 16:15 do dia 16/09/2020 às 13:45 de 12/02/2021, foi destinado ao treino dos modelos, e 30%, de 14:00 do dia 12/02/2021 até 08:30 de 04/05/2021, para teste (5812 amostras).

Figura 16 – Consumo de energia sem *outliers* - CPD1



Já no caso dos modelos de curva de carga, o dado foi particionado de maneira randômica, preservando também a proporção de 70% para treino e 30% para teste.

4.4 Etapa 4: Escolha e treinamento dos modelos

Nesta etapa, tendo em vista o referencial bibliográfico citado no Capítulo 2, cinco modelos foram escolhidos para serem testados, sendo dois para a predição do consumo energético do próximo período (ARIMA e LSTM) e três para a geração das curvas de carga (regressão linear, *SVR* e *XGBoost*). As Seções 4.4.1 e 4.4.2 tratam do processo de treinamento de cada um em detalhe.

4.4.1 Modelos para predição de consumo energético

Apesar de serem modelos capazes de realizar predições de séries temporais, ARIMA e LSTM apresentam particularidades e vantagens e desvantagens que devem ser ponderadas e analisadas com mais detalhe.

O primeiro, por se tratar de um modelo puramente linear, exige menos recursos computacionais para ser treinado. Além disso, a sua configuração, ou seja, a escolha dos seus parâmetros, é mais simples. Porém, a simplicidade do ARIMA também traz algumas desvantagens. Uma delas é a impossibilidade da utilização de um dado composto de mais de uma variável (multivariado) em seu treinamento. Dessa forma, não é possível, por exemplo, usar como entrada do modelo dados sobre a temperatura e a umidade em conjunto às medidas de consumo energético. Outro ponto negativo se origina da própria

natureza linear do ARIMA, que pode não ser capaz de capturar características não lineares contidas no dado de treinamento.

Já a LSTM, que se trata de uma rede neural, apresenta um conjunto de parâmetros internos maior e componentes não lineares em suas equações. Nesse sentido, seu treinamento permite a generalização de um maior número de funções matemáticas e, teoricamente, a capacidade de ter uma performance melhor do que o ARIMA se tratando de séries temporais. Outra vantagem da LSTM é a possibilidade de utilizar dados multivariados como entrada. Apesar disso, sua necessidade de mais recursos computacionais e sua baixa explicabilidade (redes neurais são geralmente tratadas como "caixas pretas") são uma desvantagem em relação à primeira opção.

Apesar dessas diferenças, a abordagem de treinamento será essencialmente a mesma para os dois. Primeiramente, é feita a configuração de cada um e a escolha de hiperparâmetros dos modelos. Em seguida, o modelo é treinado combinando uma técnica chamada de "origem de previsão deslizante"(ou *rolling forecasting origin*, em inglês) (HYNDMAN; ATHANASOPOULOS, 2018) e o conceito de aprendizado contínuo, em que o modelo se adapta periodicamente por meio do seu retreinamento para se adequar a novas variações e padrões no dado. Esse método funciona da seguinte forma:

1. Treina-se o modelo inicialmente com amostras entre os instantes t_a e t_b, onde $a = 0$ inicialmente;
2. Realiza-se a predição do valor para o instante $t_b + 1$ e a salva;
3. Adiciona-se a amostra do instante $t_b + 1$ no dado de treino;
4. Incrementa-se o valor de a e b em 1;
5. Repete-se do passo 1 ao 4 até i ser igual o número de amostras.

Dessa maneira, no caso do dado do CPD1, para cada nova medida existente, o modelo é treinado com dados dos últimos 140 dias, aproximadamente. Com isso, pretende-se gerar previsões mais próximas do esperado, evitando problemas relacionados a variações que ocorrem nos padrões de consumo ao longo do tempo, entre meses ou anos diferentes.

Portanto, nas seções seguintes, é explicado como foi feito o treinamento dos dois modelos. Por haver diferenças entre a configuração e passo a passo dessa etapa, há uma seção detalhando o processo utilizado para cada um.

4.4.1.1 Configuração e treinamento do ARIMA

Antes do treinamento do ARIMA, é necessário a escolha dos seus parâmetros (p, d e q). Para isso, foram analisados os gráficos ACF e PACF, mostrados nas Figuras 17 e 18

Figura 17 – Autocorrelação dos dados do CPD1

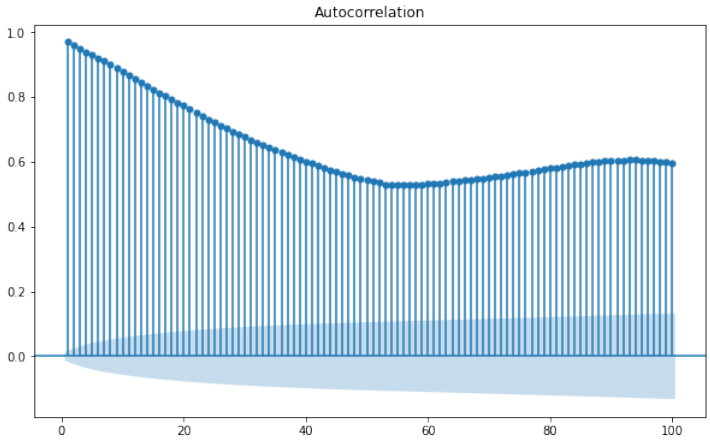
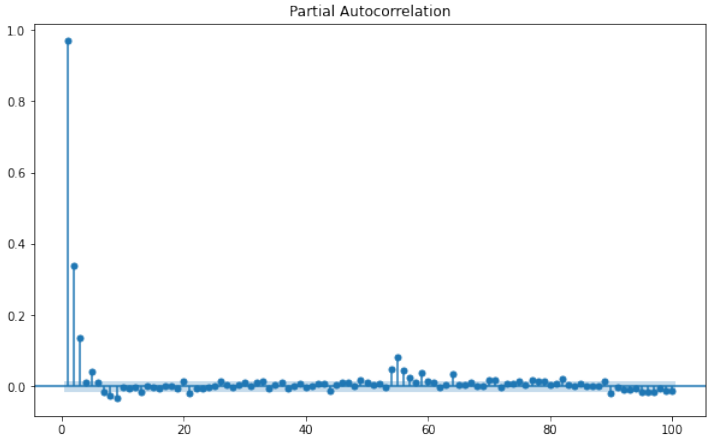


Figura 18 – Autocorrelação parcial dos dados do CPD1



Percebe-se que o gráfico de autocorrelação possui um padrão sinusoidal, ao passo que o gráfico PACF um pico no terceiro valor e mais nenhum pico significante após isso. Com isso, é indicado escolher os parâmetros com os seguintes valores: $p = 3$, $d = 0$ e $q = 0$.

Além disso, como detalhado na Seção 4.1, é possível concluir que o dado apresenta sazonalidade. Nesse contexto, adicionado à avaliação dos gráficos ACF e PACF, os melhores valores para os parâmetros da parte sazonal seriam $P = 3$, $D = 0$, $Q = 0$ e $m = 96$.

Dessa forma, a configuração que supostamente gera melhores resultados, tendo em vista o dado do CPD1, consiste em um $ARIMA(3, 0, 0)(3, 0, 0)_{96}$. Apesar de ter ciência desse fato, foi feita a escolha pelo uso apenas da parte não-sazonal desse modelo. Isso se deu por dois fatores: primeiramente, a necessidade de recursos computacionais para otimizar os coeficientes internos do modelo considerando um valor de $m = 96$ é muito superior ao que está acessível para a realização dos experimentos desse trabalho no presente momento. Essa condição também impacta o requisito da necessidade de retreinamento contínuo do modelo, que deve ocorrer de maneira eficiente e célere.

Em segundo lugar, modelos ARIMA geralmente são projetados para terem valores menores de m, como 4 para dados trimestrais ou 12 para dados anuais. Dessa maneira, apesar de ser possível utilizar teoricamente qualquer valor de m, a realização de uma diferenciação sazonal de ordem muito alta, como por exemplo comparar o que ocorreu na medição atual com 96 medições atrás, não faz muito sentido. (HYNDMAN, 2010)

Por último, também foi realizado o teste estatístico de Ljung-Box, explicado na Seção 2.3.3.7, tomando como base os resíduos produzidos pelo modelo $ARIMA(3, 0, 0)$ treinado inicialmente. O resultado do teste, exibido na Tabela 1, indica que não é possível descartar a hipótese nula (os resíduos são independentemente distribuídos, isto é, ruído branco) para atrasos de até 4 períodos. Porém, além desse valor, o p-valor do teste fica abaixo de 0.05, o que indica que, nestes casos, devemos rejeitar a hipótese nula e aceitar a hipótese de que há correlação entre os resíduos deixados pelo modelo.

Tabela 1 – Resultados do teste de Ljung-Box para resíduos do ARIMA(3, 0, 0)

Atraso	**Estatística de teste**	**p-valor**
1	0.005662	0.940021
2	0.144614	0.930245
3	1.966530	0.579383
4	9.358165	0.052745
5	20.680660	0.000931
6	26.553685	0.000176
7	29.540629	0.000115
8	38.124109	0.000007
9	39.059843	0.000011
10	43.343195	0.000004

Sendo assim, os modelos escolhidos para os experimentos são versões não-sazonais, sendo eles: $ARIMA(3, 0, 0)$ e $ARIMA(3, 1, 0)$.

4.4.1.2 Configuração e treinamento da LSTM

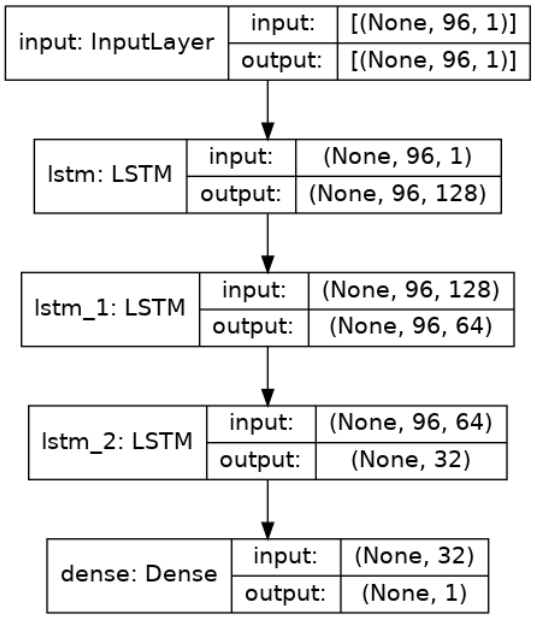
A LSTM usada consiste em um modelo simples com apenas quatro camadas, compostas conforme a Figura 19.

Antes do treinamento, é necessário a realização de uma normalização no dado. Esse passo é necessário pois os pesos iniciais da rede construída com a biblioteca *Keras* geralmente consistem em valores muito baixos (KERAS, 2021b). Para realizar isso, foi utilizado o *MinMaxScaler* da biblioteca *scikit-learn* (SCIKITLEARN, 2021a).

Esse método é bem simples e permite escalonar os valores originais para uma faixa entre zero e um. Para isso, para cada x_i em um conjunto X, tem-se:

$$x_i = \frac{(x_i - min(X))}{(max(X) - min(X))} \tag{4.1}$$

Figura 19 – Arquitetura da LSTM utilizada



Com base no dado normalizado e estruturado em janelas compostas de 96 valores, equivalente a 1 dia de medições, o modelo foi treinado em duas etapas: uma inicial e a formada pelo treinamento contínuo. O primeiro passo contou com os 70% mais antigos do dado e foi constituído por 15 épocas com um tamanho de *batch* de 32.

Já o treinamento contínuo e a geração das predições seguiu uma metodologia sutilmente diferente da aplicada utilizando o ARIMA. No caso da LSTM, por se tratar de uma rede neural, o modelo treinado inicialmente não tem os pesos aprendidos sobrescritos. Dessa forma, para cada nova amostra dos 30% finais do dado, é feito apenas um novo ajuste da rede para um *batch* de uma amostra apenas, ao invés de se selecionar um período entre t_a e t_b.

4.4.2 Modelos para geração de curvas de carga

Como citado anteriormente, três modelos foram utilizados nos experimentos de geração de curvas de carga: regressão linear, *SVR* e *XGBoost*. Apesar de possuírem estratégias bem diferentes em relação ao cálculo da regressão, já detalhadas na Seção 2.3.1.1, o processo de treinamento foi idêntico para todos eles, sendo composto de:

1. Leitura dos dados do sensor CPD1, para o período de 16/09/2020 a 12/02/2021;
2. Remoção de *outliers*;
3. Separação "randômica"do dado em treino/teste, respeitando a proporção de 70/30, utilizando sempre a mesma *seed*, o que faz com que as partições contenham as mesmas amostras para todos os experimentos;

4. Treinamento do modelo, empregando sempre os valores *default* atribuídos pelas bibliotecas para os parâmetros; [3]

5. Avaliação dos modelos, utilizando a métrica *Mean Absolute Error* (MAE);

6. Geração da curva de carga para 24 horas.

4.5 Etapa 5: Avaliação dos modelos

Todos os modelos gerados segundo os processos explicados na seção anterior foram avaliados utilizando o erro absoluto médio, dado por (SCIKITLEARN, 2021a):

$$MAE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i| \tag{4.2}$$

Além disso, foram feitas análises gráficas comparando os valores do dado de teste e os valores preditos.

4.5.1 Avaliação dos modelos para predição de série temporal

Inicialmente, se avaliou o MAE (*Mean Absolute Error*) para os modelos testados, mostrado na Tabela 2.

Tabela 2 – MAE para predições dos modelos testados

Modelo	**MAE**
ARIMA(3, 0, 0)	89.593
ARIMA(3, 1, 0)	89.356
LSTM	88.154

Também foi registrado o tempo de treinamento para cada um, o que é exibido nas Tabela 3.

Tabela 3 – Tempo de treinamento dos modelos testados (predição de série temporal)

Modelo	**Tempo de treinamento (HH:MM:SS)**
ARIMA(3, 0, 0)	03:00:14
ARIMA(3, 1, 0)	01:33:49
LSTM	00:48:12

Além disso, nas Figuras 20, 21 e 22, é possível ver a comparação entre o dado de teste e o dado predito pelos modelos.

[3] Documentação do scikit-learn: SVR e LinearRegression.
Documentação do xgboost: XGBoost Parameters

Figura 20 – Resultado do ARIMA(3, 0, 0)

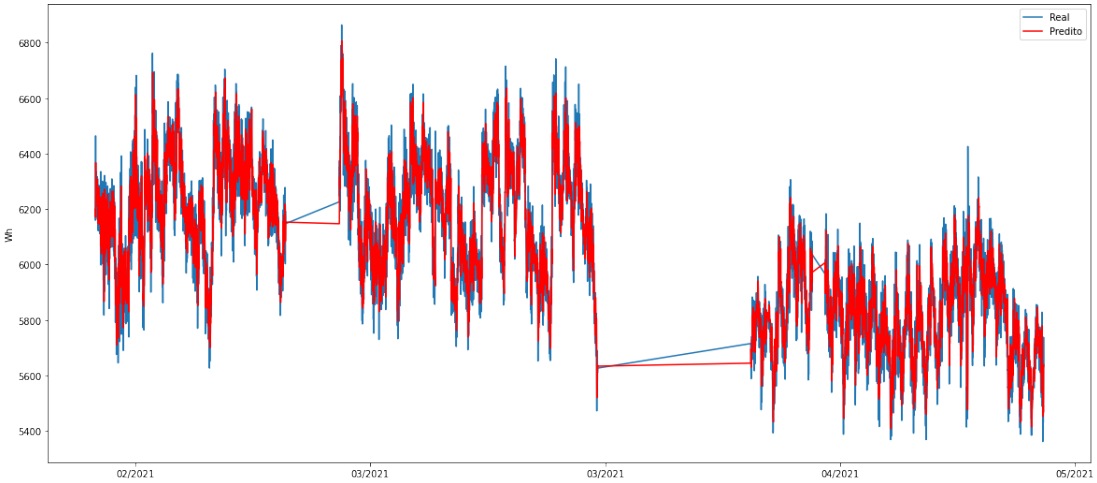


Figura 21 – Resultado do ARIMA(3, 1, 0)

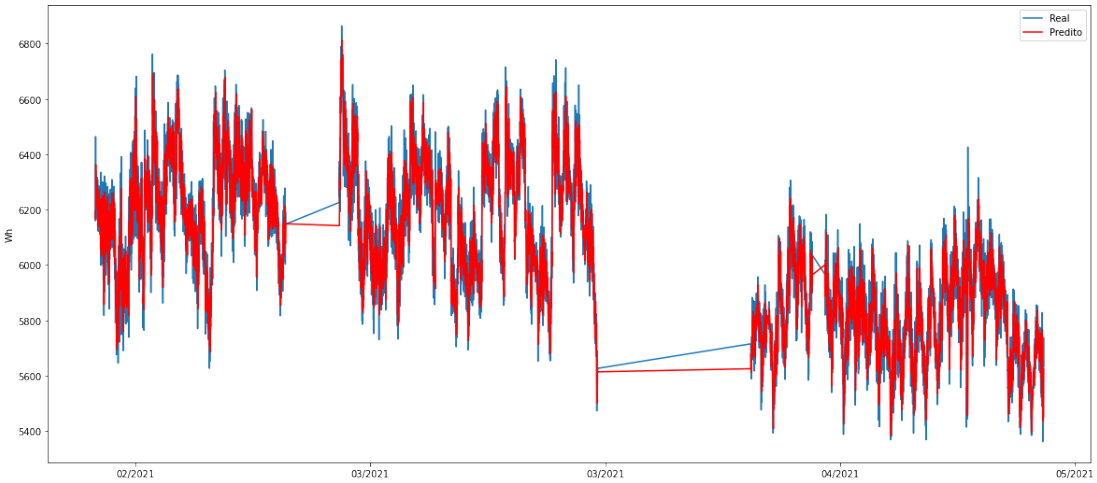
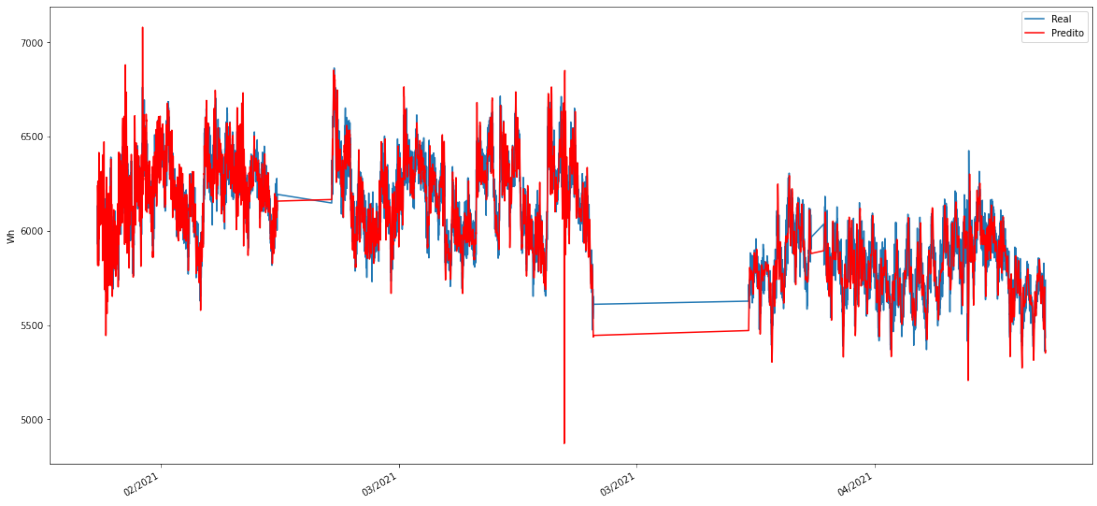


Figura 22 – Resultado da LSTM



Outro tipo de avaliação realizada é a comparação entre os resultados tomando como base apenas um subconjunto do dado de teste equivalente a um dia (96 medições),

como mostrado nas Figuras 23, 24 e 25

Figura 23 – Resultado do ARIMA(3,0,0) para um dia

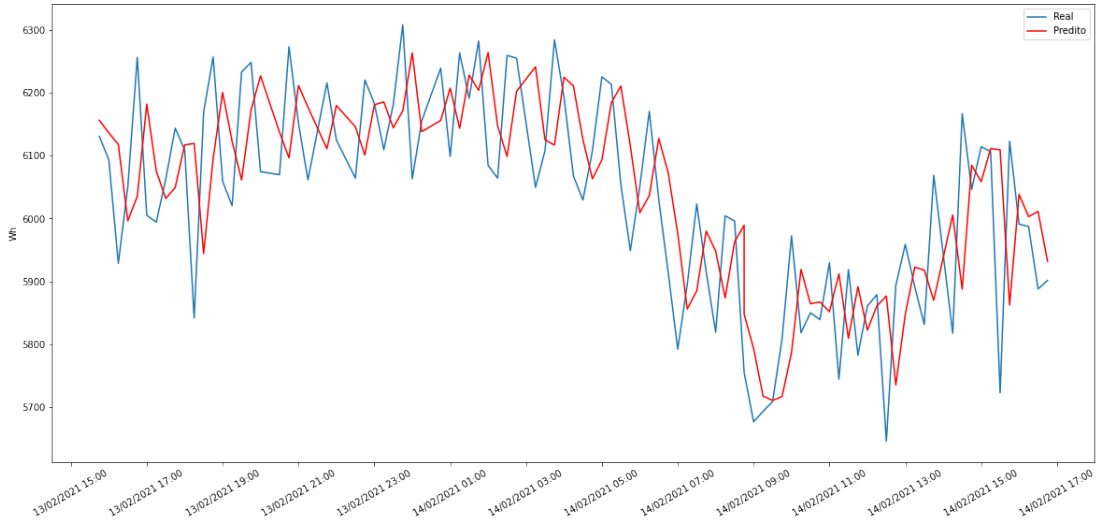
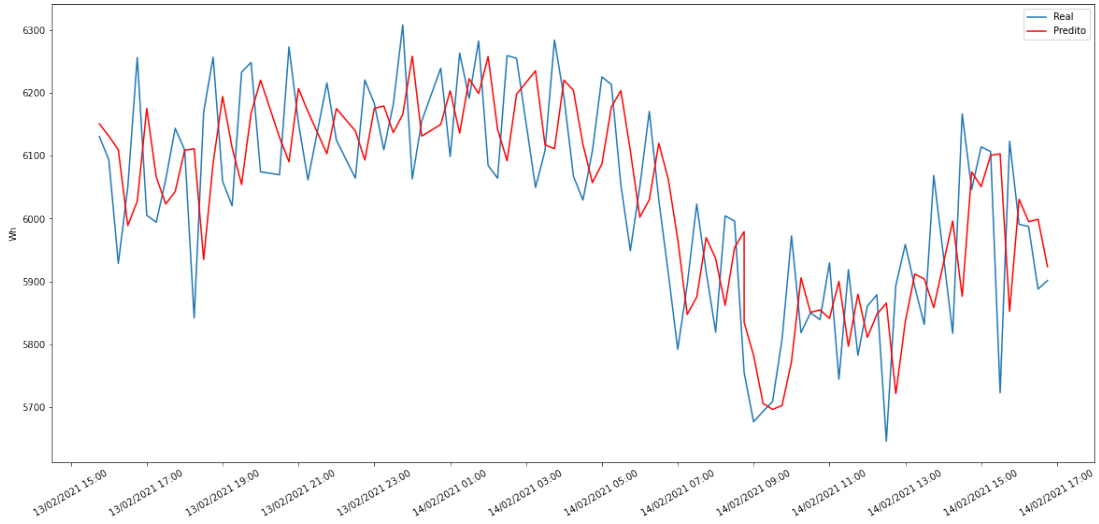


Figura 24 – Resultado do ARIMA(3,1,0) para um dia



Por fim, foi feita uma comparação para o corte de um dia entre as duas versões treinadas para o ARIMA, mostrado na Figura 26.

4.5.2 Avaliação dos modelos de geração de curvas de carga

Para os modelos de curva de carga, também se avaliou a métrica MAE e os tempos de treinamento de cada um, observados nas Tabelas 4 e 5, respectivamente.

Ademais, também foram geradas as respectivas curvas de carga para cada modelo e a comparação das três, exibido nas Figuras 27, 28, 29 e 30.

Figura 25 – Resultado da LSTM para um dia

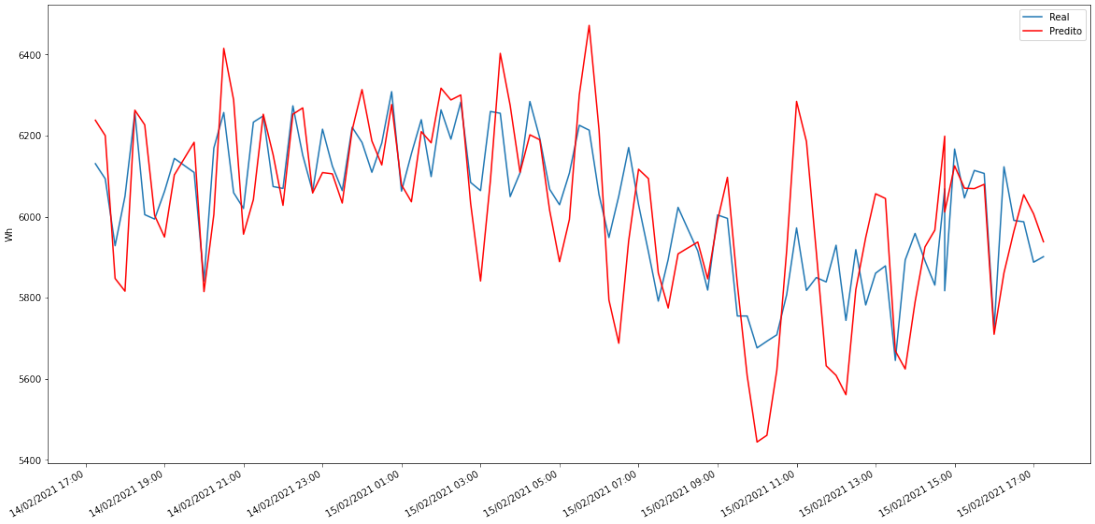


Figura 26 – Comparação entre ARIMA(3, 0, 0) e ARIMA(3, 1, 0) para um dia

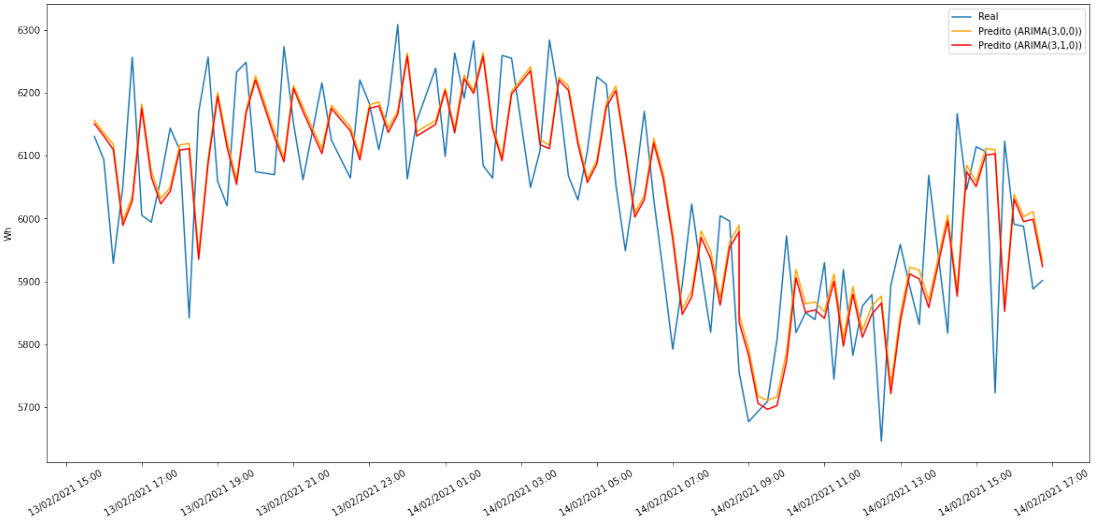


Tabela 4 – MAE para predições dos modelos testados (geração de curvas de carga)

Modelo	**MAE**
Regressão linear	399.453
SVR	378.593
XGBoost	392.100

Tabela 5 – Tempo de treinamento dos modelos testados (geração de curvas de carga)

Modelo	**Tempo de treinamento (HH:MM:SS)**
Regressão linear	00:00:17
SVR	00:00:31
XGBoost	00:00:44

Figura 27 – Curva de carga (Regressão linear)

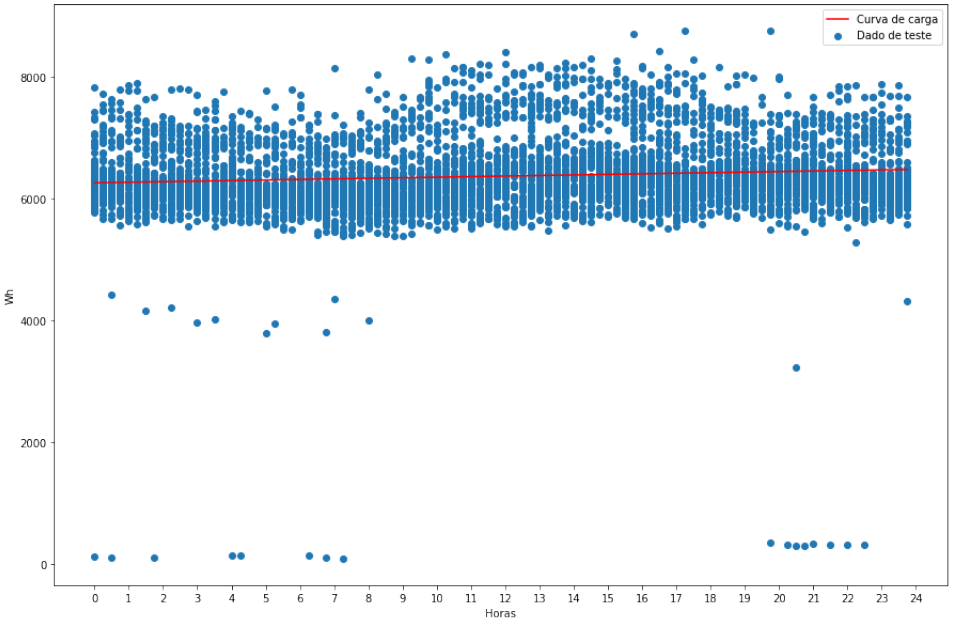


Figura 28 – Curva de carga (SVR)

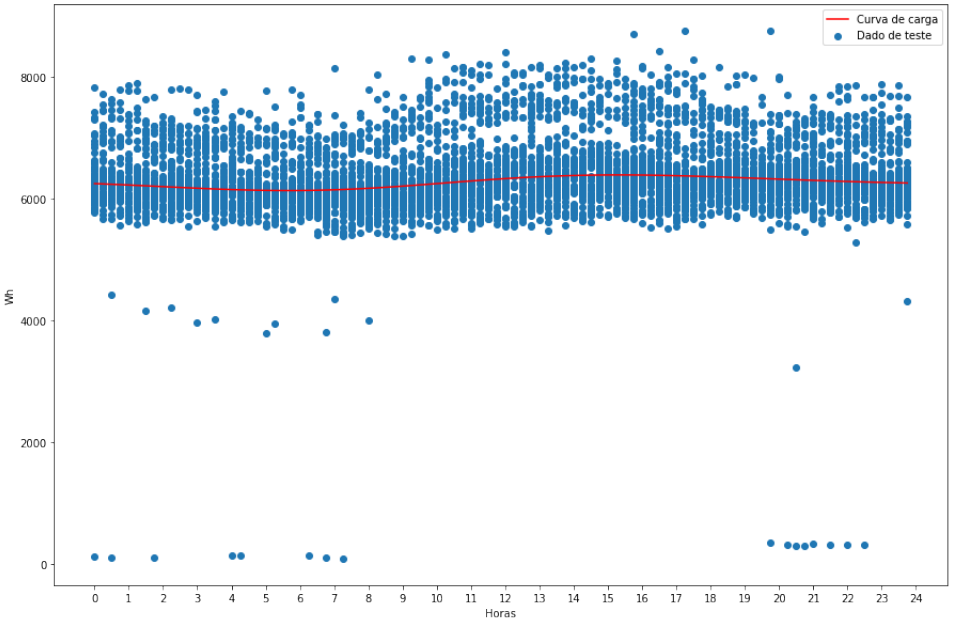


Figura 29 – Curva de carga (XGBoost)

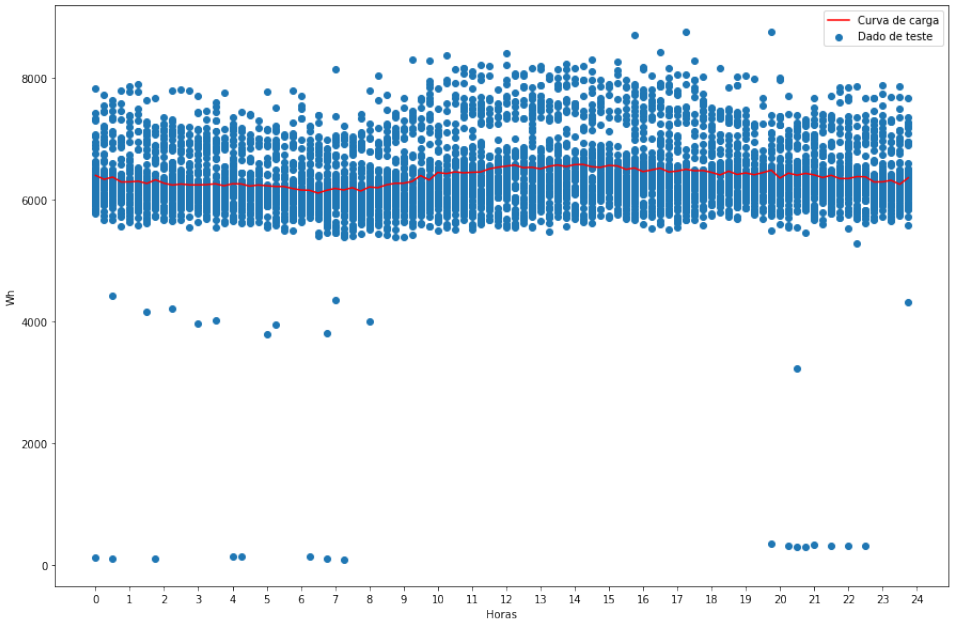
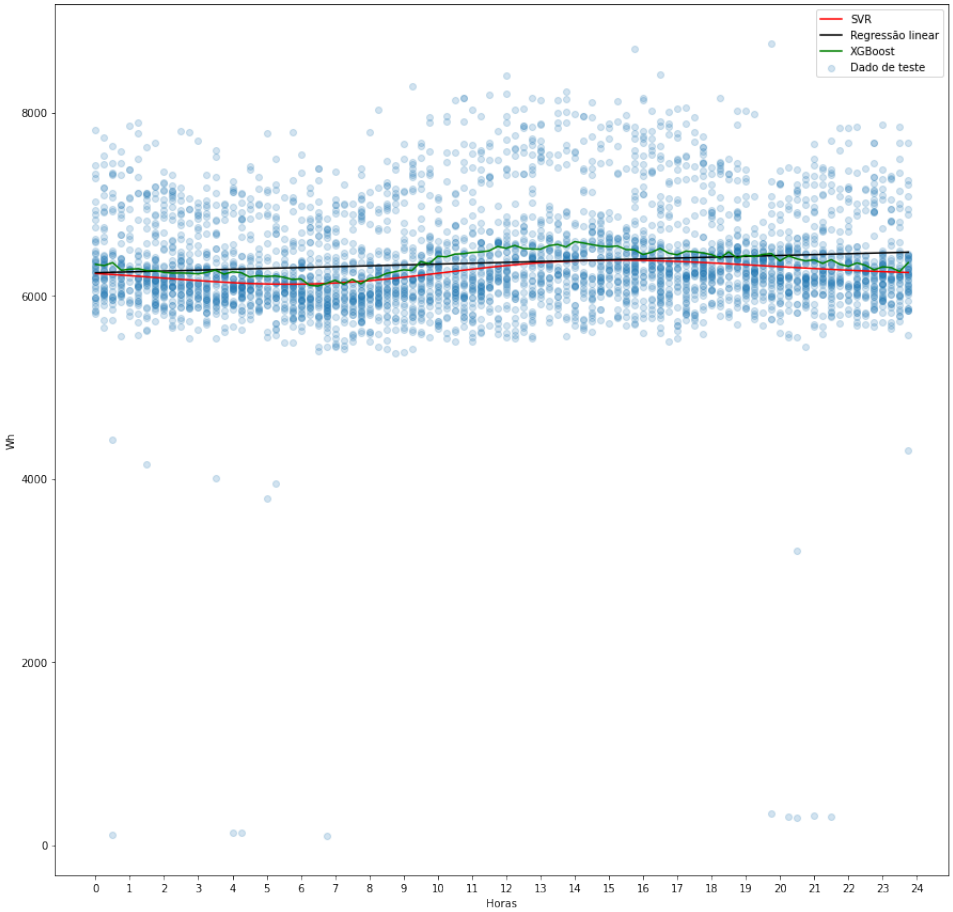


Figura 30 – Comparação das curvas de carga geradas



5 SIGEML

Este capítulo abrange os requisitos, a metodologia, a arquitetura e os resultados do sistema desenvolvido conforme os objetivos deste trabalho, definidos na Seção 1.2. Esse sistema, nomeado como SIGEML, tem como principais requisitos, que serão melhor especificados, os seguintes:

- **Comunicação com API do SIGE**: o sistema deve ser capaz de consumir dos dados gerados em tempo real pela plataforma SIGE através dos *endpoints* disponibilizados;
- **Geração de curvas de carga**: para fins de estudo e exploração, o sistema deve permitir que os usuários gerem curvas de carga baseado nos dados consumidos da API do SIGE para períodos de tempo selecionados;
- **Previsão do consumo de energia**: o sistema deverá ser capaz de gerar o valor previsto para o consumo de energia para o instante da próxima medição realizada pelo SIGE;
- **Interface para o usuário**: o sistema deverá prover uma interface que permita a visualização das curvas de carga previstas e dos valores de consumo previstos em tempo real, por meio de uma experiência de usuário intuitiva;
- **Interface para outros sistemas**: o sistema deverá permitir que outros sistemas, incluindo o próprio SIGE, utilizem dos serviços disponibilizados por meio de uma API.

5.1 Requisitos do sistema

Em uma fase anterior às decisões arquiteturais e ao desenvolvimento da ferramenta em si, foi realizada uma etapa de elicitação de requisitos. Esse estágio teve como objetivo definir com um maior detalhe os elementos considerados relevantes ou essenciais do sistema a ser construído, tomando como base os cinco aspectos mais abrangentes apresentados no início deste Capítulo 5.

A partir do estabelecimento desse rol de requisitos com uma menor granularidade, viabilizou-se a priorização dos mesmos. Essa etapa foi fundamentada na técnica *MoSCoW*, uma abordagem colaborativa, originada do *Dynamic Software Development Method* (DSDM), que designa uma entre quatro categorias de importância para cada requisito (HUDAIB et al., 2018):

- *Must have* ("Deve ter"): nível dado a um requisito crítico para o projeto;
- *Should have* ("Deveria ter"): representa uma *feature* de alta prioridade, mas não crítica para o projeto;
- *Could have* ("Poderia ter"): simboliza um requisito desejável, mas não necessário;
- *Won't have* ("Não terá"): configura um requisito que não será implementado na etapa atual do projeto, mas que futuramente pode ser integrado a ele.

Com base nesse método e considerando os requisitos previamente estabelecidos para o SIGEML, foi construída a Tabela 6.

Tabela 6 – Priorização dos requisitos.

Código	**Requisito**	**Prioridade**
R01	O usuário deve ser capaz de criar um novo experimento que gera uma curva de carga	MUST
R02	O usuário deve ser capaz de visualizar os experimentos criados	MUST
R03	O usuário deve ser capaz de visualizar o resultado com as métricas do experimento	MUST
R04	O usuário deve ser capaz de visualizar o tempo de duração do experimento	MUST
R05	O usuário deve ser capaz de visualizar os parâmetros de configuração do experimento	MUST
R06	O usuário deve ser capaz de remover experimentos	MUST
R07	O usuário deve ser capaz de selecionar o conjunto de dados para a execução de um novo experimento	MUST
R08	O usuário deve ser capaz de selecionar os parâmetros do modelo de um novo experimento	MUST
R09	O usuário deve ser capaz de filtrar experimentos por nome	SHOULD
R10	O usuário deve ser capaz de filtrar experimentos por status	COULD
R11	O usuário deve ser capaz de filtrar experimentos por tipo de modelo	SHOULD
R12	O usuário deve ser capaz de filtrar experimentos por data de criação	COULD
R13	O usuário deve ser capaz de comparar curvas de carga de experimentos distintos	SHOULD
R14	O usuário deve ser capaz de visualizar predições de consumo energético em tempo real	COULD

R15	O usuário poderá selecionar o número de períodos retornados pela predição em tempo real	COULD
R16	O treinamento de um novo modelo deve ocorrer de modo assíncrono	SHOULD
R17	O sistema deve ser capaz de exportar modelos salvos	COULD
R18	O sistema deve possibilitar gerenciamento de usuários	WON'T
R19	O usuário pode ordenar experimentos por nome	COULD
R20	O usuário pode ordenar experimentos por data de criação	COULD
R21	O usuário deve ser capaz de realizar treinamentos configuráveis da LSTM	WON'T

5.2 Metodologia de desenvolvimento

A metodologia de desenvolvimento de software utilizada na construção do SIGEML foi o *Scrum*, que pertence à família de métodos ágeis de software. De acordo com SUTHERLAND; SCHWABER, como definido no *The Scrum Guide*, o *Scrum* emprega uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar o risco.

Dessa maneira, por meio da combinação de quatro eventos formais para inspeção e adaptação dentro de um evento contido, nomeado como *Sprint*, são implementados os pilares empíricos do *Scrum* de transparência, inspeção e adaptação. A seguir, serão melhor esclarecidos os papéis, artefatos, e eventos do *Scrum* utilizados no desenvolvimento no SIGEML.

5.2.1 Papéis

- ***Stakeholders***: partes interessadas no desenvolvimento do projeto que podem ajudar na definição de requisitos e planejamento do *product backlog*. Um dos principais *stakeholders* do projeto, por exemplo, são os Professores Doutores Renato Coral Sampaio e Alex Reis;
- **Desenvolvedores**: responsáveis pelo cumprimento do *sprint backlog* permitindo assim uma evolução incremental do produto. Além disso, podem definem padrões de qualidade de entrega e auxiliam na construção e priorização do *sprint backlog*. Os autores deste trabalho de conclusão assumiram papéis de desenvolvedores.

5.2.2 Artefatos

- ***Product Backlog***: uma lista ordenada do que é necessário para incrementar o produto. Essa lista evolui ao longo do desenvolvimento do projeto;

- ***Sprint Backlog***: conjunto de itens escolhidos do *product backlog* que serão entregues ao longo da *sprint* para a qual foram selecionados.

5.2.3 Eventos

- ***Sprint***: como principal evento da metodologia *scrum*, a *sprint* tem duração fixa de no máximo 1 mês e se caracteriza pela presença de eventos executados internamente a uma *sprint*. É na *sprint* que ideias são transformadas em valor de produto;
- ***Sprint Planning***: evento que inicia a *sprint* definindo o trabalho a ser executado durante a *sprint*. Todos do time podem participar desse planejamento;
- ***Sprint Review***: evento que finaliza a *sprint* no qual o time revisa o que foi feito e levanta pontos de melhoria para a próxima *sprint*.

A Tabela 7 mostra em detalhes a distribuição do desenvolvimento dos requisitos definidos no *product backlog* da Tabela 6 ao longo das sete *sprints*. A tabela também esclarece o *sprint backlog* exibindo as tarefas que foram executadas em cada *sprint*.

Tabela 7 – Mapeamento das *sprints*, requisitos e tarefas no desenvolvimento do SIGEML.

Sprint	**R**	**Tarefas**	**TC**	**RC**
Sprint 1 02/08/21 - 15/08/21		Configurar projeto do Back-end	Sim	
	R04	Configurar MLflow no back-end	Sim	Sim
		Configurar projeto do Front-end	Sim	
Sprint 2 16/08/21 - 29/08/21	R02	Adicionar página com lista de experimentos	Sim	Sim
		Adicionar paginação no FastAPI	Sim	
	R04	Configurar MLflow no back-end	Sim	Sim
	R09	Adicionar busca de experimento pelo nome	Sim	Sim
Sprint 3 30/08/21 - 12/09/21	R03	Adicionar exibição de detalhes do experimento	Sim	Não
		Adicionar métricas na modal do experimento	Sim	
		Implementar repositório de modelos no back-end	Sim	
		Adicionar gráfico com resultado na modal do experimento	Não	
	R05	Adicionar configuração na modal do experimento	Sim	Sim
	R10	Adicionar campo para filtrar por status	Não	Não
Sprint 4 13/09/21 - 26/09/21	R03	Adicionar gráfico com resultado na modal do experimento	Sim	Sim
	R10	Adicionar campo para filtrar por status	Sim	Sim
	R11	Realizar filtragem no back-end	Sim	Sim
		Adicionar campo para filtrar por tipo de modelo	Sim	

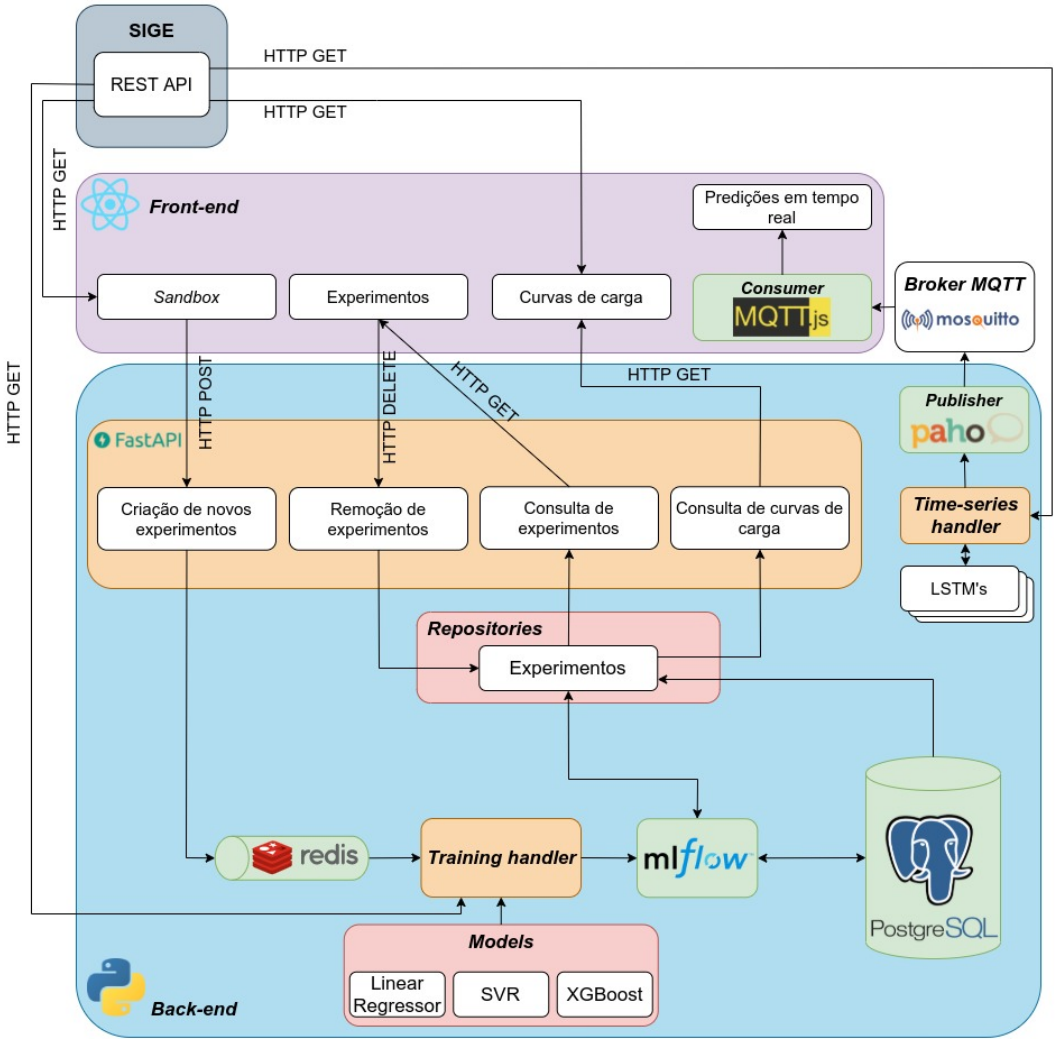
	R01	Adicionar tela para sandbox	Sim	Sim
		Adiconar formulário para criar experimento	Sim	
Sprint 5 27/09/21 - 10/10/21	R08	Adicionar etapas de escolha do modelo e parâmetros do experimento a ser criado	Sim	Sim
	R07	Fazer requisição para o SIGE através de campos de entrada	Sim	Sim
		Mostrar resultado da requisição de dados de consumo do SIGE em uma tabela	Sim	
	R06	Criar endpoint para excluir experimento	Sim	Sim
		Adicionar botão de excluir para cada experimento listado na página de experimentos	Sim	
	R16	Adicionar serviço do redis e implementar training handler no backend	Sim	Sim
Sprint 6 11/10/21 - 24/10/21	R13	Armazenar curvas de carga no banco de dados	Sim	Sim
		Adicionar página para comparação de curvas de carga entre experimentos	Sim	
		Adicionar campos de prédio, data de início e fim, e experimentos que serão comparados	Sim	
		Implemetar requisições para API do SIGE	Sim	
		Construir gráfico com pontos de teste e predições feitas pelos modelos dos experimentos selecionados	Sim	
Sprint 7 25/10/21 - 07/11/21	R14	Configurar serviço do broker MQTT no back-end	Sim	Sim
		Implementar time-series handler para consultar os dados do SIGE periodicamente e realizar predições de consumo através da LSTM	Sim	
		Construir página de predições em tempo real	Sim	
		Adicionar campo com seleção de prédio	Sim	
		Consumir dados de consumo e predições do broker MQTT	Sim	
		Construir gráfico em tempo real com os dados provenientes do broker MQTT	Sim	

Legenda: R - Requisito(s), TC - Tarefa Concluída, RC - Requisito Concluído.

5.3 Arquitetura do sistema

A concepção da arquitetura do SIGEML se baseou em duas premissas básicas: a existência de uma interface gráfica que fosse intuitiva e permitisse o treinamento e experimentação com modelos para predição de curvas de carga e a presença de uma API que pudesse ser consumida por outros serviços de maneira independente. Assim, com base nesses pontos, foi pensado a estrutura exibida na Figura 31.

Figura 31 – Arquitetura do SIGEML



5.3.1 *Back-end*

O *back-end* do SIGEML [1], desenvolvido em *Python* (PYTHON, 2021), é composto essencialmente por:

- Uma API construída com *FastAPI*, responsável por disponibilizar a criação, consulta e deleção de experimentos;

[1] O repositório com o código fonte do *back-end* do SIGEML pode ser encontrado aqui

- Repositórios de experimentos. Um repositório é uma camada de abstração, usualmente uma classe ou componente, responsável por encapsular a lógica requerida para acessar uma fonte de dados;
- Uma fila do *Redis*, utilizada para receber eventos de treinamento de novos modelos enviados por um *endpoint* da API;
- O *training handler*, que é o serviço responsável por lidar com os novos eventos de treinamento retirados da fila, com a instanciação do respectivo modelo requerido e o disparo de seu treinamento via *mlflow*;
- *Models*, que tratam-se de classes que abstraem os objetos e métodos originais dos modelos disponibilizados pelas bibliotecas utilizadas;
- *mlflow*, que gerencia o ciclo de vida dos experimentos criados e o armazenamento de seus metadados no banco de dados;
- *PostgreSQL*, o sistema gerenciador do banco de dados utilizado;
- O *time-series handler*, serviço encarregado de periodicamente requisitar à API do SIGE os valores de consumo atuais para os prédios disponíveis, retreinar as LSTM's e publicar as predições via *broker* MQTT;

Nesse sentido, é possível definir dois fluxos principais que compõem o funcionamento do *back-end*. De um lado, há o contexto de geração e consulta de curvas de carga, com a API agindo como uma interface para essas ações; de outro, o processamento e envio das predições em tempo real.

5.3.1.1 Geração e consulta de experimentos de curvas de carga

Um dos pontos principais do *back-end* do SIGEML é fornecer uma interface via API que permita a criação e gerenciamento de experimentos com curvas de carga. Um "experimento", no caso deste trabalho específico, trata-se da execução de um treinamento de um modelo utilizando parâmetros específicos e um conjunto de dados pré-selecionado.

Cada treinamento desse, que é chamado de *run* pela biblioteca *mlflow* (MLFLOW, 2021), ao final, gera e persiste, entre outras informações:

- Um identificador único;
- Os parâmetros utilizados no treinamento;
- O valor da métrica para as predições feitas pelo modelo treinado;
- A curva de carga resultante;

- A instância do modelo treinado (opcionalmente).

Por meio desses dados, é possível avaliar e comparar os resultados dos testes feitos, assim como realizar consultas no histórico das execuções passadas, tudo por meio de requisições HTTP para a API disponibilizada. Para realizar **criação de novos experimentos**, por exemplo, tem-se um fluxo de execução formado por quatro etapas:

1. Uma requisição HTTP POST é feita para o *endpoint* de treinamento contendo um *payload* com configurações gerais e parâmetros do modelo e do dado que serão utilizados no treinamento;
2. Um evento para esse *payload* é criado e inserido um uma fila do Redis;
3. Um método em *Python*, que é executado como um serviço, retira esse novo evento da fila e aciona o *mflow* instanciando e treinando o modelo requisitado;
4. Os parâmetros, métricas e a curva de carga correspondente ao experimento são armazenadas no PostgreSQL pelo *mlflow*.

Já para a **remoção de experimentos**, pode-se considerar um fluxo constituído assim:

1. Uma requisição HTTP DELETE é feita para o *endpoint* de deleção de experimento contendo o identificador único da "*run*"(execução) que se deseja apagar;
2. O *endpoint* chama o método de deleção do repositório de experimentos;
3. No repositório, o acesso aos registros do banco é feito por chamadas à API do *mlflow*, que lida diretamente com o armazenamento em si dos experimentos e modelos criados. Por meio da API, a execução salva de um experimento pode ser excluída.

No caso da **consulta de experimentos e de curvas de carga**, tem-se um passo a passo similar. Para o primeiro caso, ocorre o seguinte:

1. Uma requisição HTTP GET é feita para o *endpoint* de listagem de experimentos;
2. O *endpoint* chama o método do repositório que consulta e retorna a lista de experimentos com base nos filtros selecionados.

Já para o segundo, que tem a função de consultar o e retornar as curvas de carga resultantes dos experimentos, o fluxo ocorre assim:

1. Uma requisição HTTP GET é feita para o *endpoint* de listagem de experimentos, passando o nome dos experimentos desejados para comparação;

2. O *endpoint* chama o método do repositório que consulta e retorna a lista de experimentos com base nos nomes selecionados. [2]

5.3.1.1.1 Configuração de novos experimentos de curva de carga

Como explicado na seção anterior, ao criar um novo experimento, é permitido ao usuário configurar determinados aspectos do dado e do modelo utilizados no treinamento. Entre as opções genéricas, ou seja, independentes de modelo, tem-se:

- O nome do experimento;
- O nome do modelo a ser treinado;
- O tamanho do *dataset* de teste utilizado;
- Se o experimento deve armazenar uma instância do modelo treinado no banco de dados.

Além disso, há como determinar o valor para alguns parâmetros dos modelos disponíveis. São eles:

- Regressão linear (SCIKITLEARN, 2021c):
  - *fit_intercept*: se deve ser calculado o valor do intercepto;
  - *positive*: se deve forçar que os coeficientes sejam positivos.
- SVR (SCIKITLEARN, 2021d):
  - *kernel*: o tipo de *kernel* a ser usado;
  - *epsilon*: o valor para o parâmetro ϵ, mostrado em 6;
  - *gamma*: coeficiente para os *kernels "rbf"*, *"poly"* e *"sigmoid"*;
  - *tol*: tolerância para o critério de parada das iterações;
  - *C*: parâmetro que determina "rigidez"na determinação das margens. Explicado em detalhe na Seção 2.3.1.1.2;
  - *max_iter*: número máximo de iterações.

[2] Como o *mlflow* não suporta armazenar objetos mais complexos como listas ou dicionários nas tabelas que gerencia, o armazenamento e consulta das curvas de carga geradas é feito de forma direta, sem interface da biblioteca.

- *XGBoost* (XGBOOST, 2021b):
  - *n_estimators*: número de árvores usadas no *gradient boosting*;
  - *max_depth*: profundidade máxima das árvores utilizadas como *weak learners*;
  - *learning_rate*: taxa de aprendizado utilizado na atualização do modelo;
  - *gamma*: valor mínimo para a função de custo para realizar uma partição adicional em um nó folha;
  - *random_state*: valor para *random seed* do modelo.

5.3.1.2 Predições em tempo real

Além das curvas de carga, foi implementado no *back-end* um componente responsável pela predição das séries temporais de consumo energético dos prédios. Seu funcionamento se dá da seguinte forma:

1. Periodicamente (considerando a configuração atual do SIGEML, a cada 15 min), o *Times-series handler* realiza uma requisição para a API do SIGE para cada prédio cadastrado;
2. Com os dados de consumo coletados, é feito um retreino das LSTM's, uma por medidor, utilizando uma janela de tempo pré-determinada que inclui o novo dado coletado;
3. É feita a predição do consumo esperado do próximo período para cada medidor e esse valores são publicados no *broker* MQTT.

5.3.2 *Front-end*

O *Front-end* do SIGEML [3] é a interface gráfica do sistema no qual o usuário poderá interagir. Desenvolvido em *ReactJS*(REACTJS, 2021), é formado por quatro telas principais:

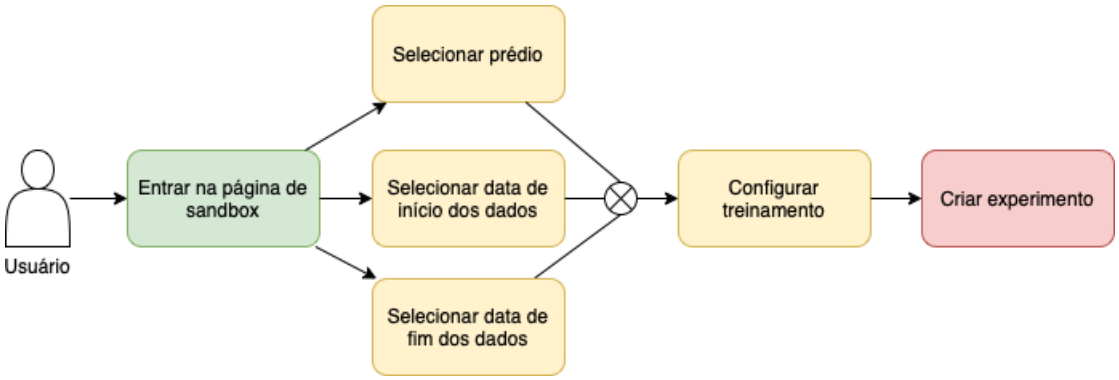
- ***Sandbox***: responsável pela criação de novos experimentos a partir de dados de consumo provenientes da API do SIGE;
- **Experimentos**: contém a lista de experimentos criados;
- **Curvas de Carga**: permite a comparação visual da performance entre experimentos distintos;
- **Predições em tempo real**: constituída de um gráfico em tempo real que exibe a previsão de consumo para os próximo quinze minutos.

[3] O repositório com o código fonte do *front-end* do SIGEML pode ser encontrado aqui

5.3.2.1 *Sandbox*

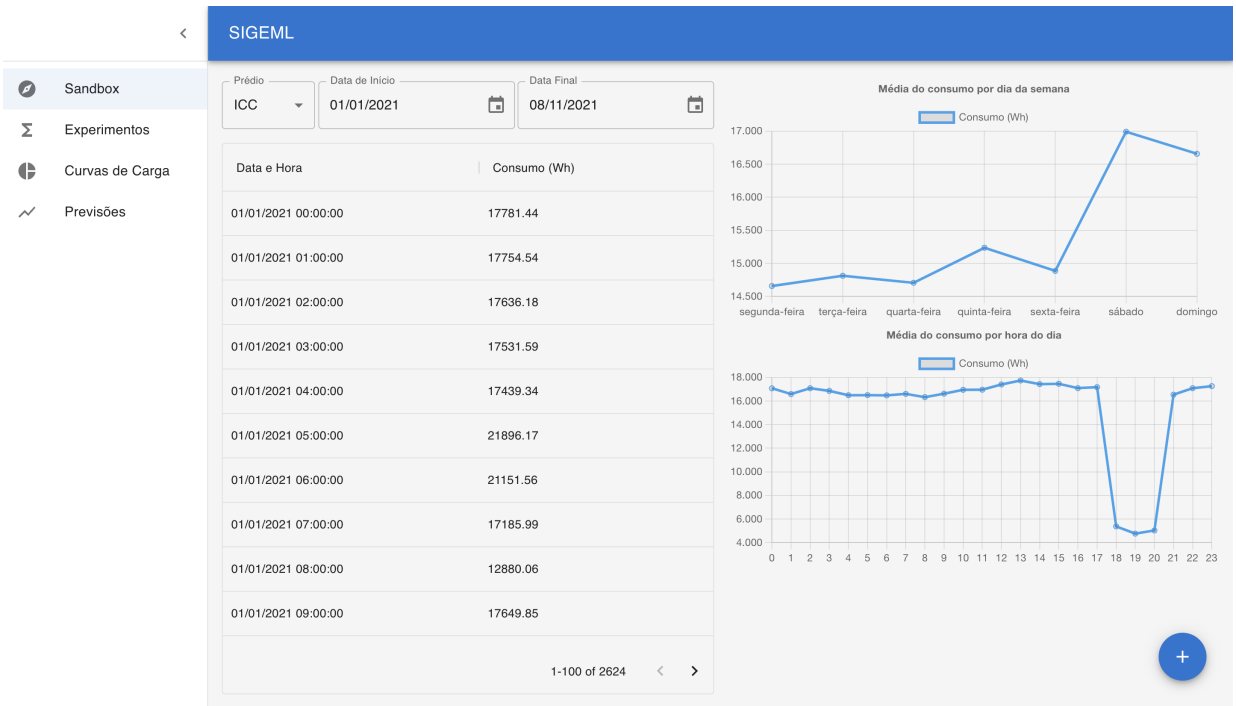
O fluxo da página de *Sandbox* pode ser representado pelo diagrama da Figura 32.

Figura 32 – Diagrama de Fluxo da página de *Sandbox*



Assim, ao entrar nesta página, o usuário seleciona o prédio e a data de início e fim dos dados coletados pelo sensores do SIGE, conforme mostrado na Figura 33.

Figura 33 – Tela do *Sandbox*



Após isso, o usuário poderá escolher um modelo e os parâmetros a serem utilizados em seu treinamento, conforme ilustrado nas Figuras 34, 35 e 36.

5.3.2.2 Experimentos

A página de experimentos possibilita a consulta dos detalhes de experimentos previamente criados. As opções disponíveis nela são exibidos na Figura 37.

Figura 34 – Tela do *Sandbox*: Formulário de seleção do modelo

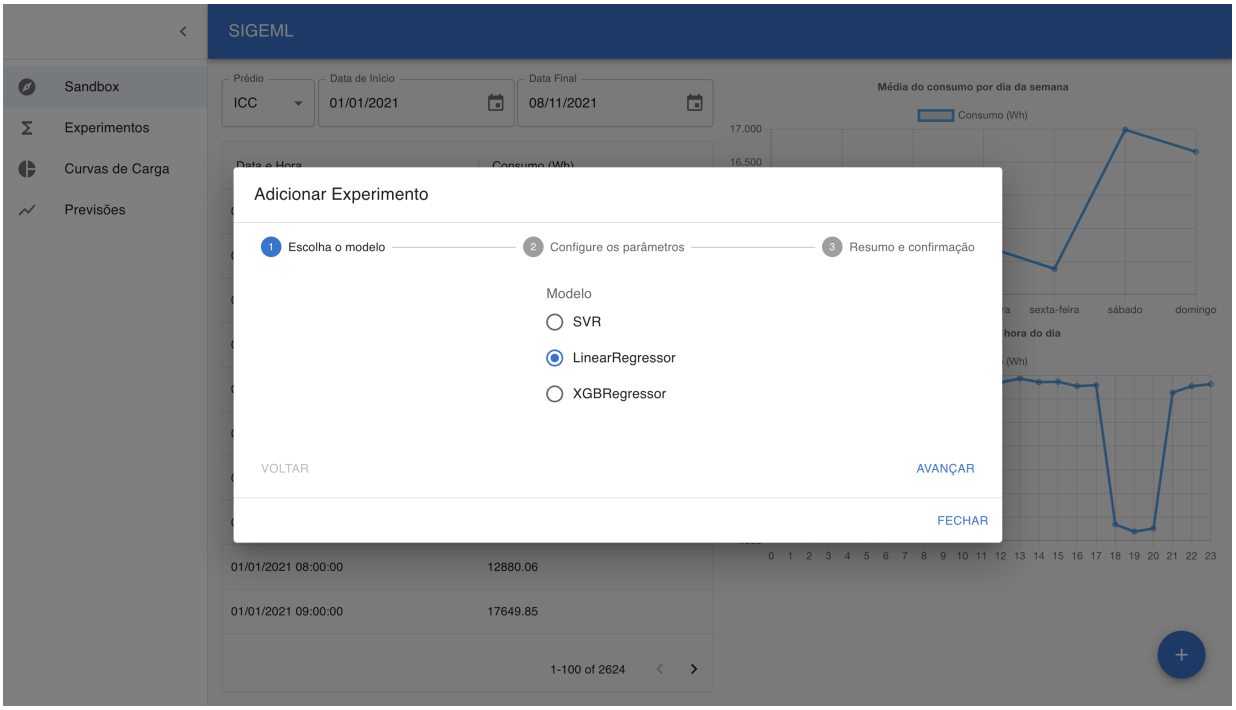
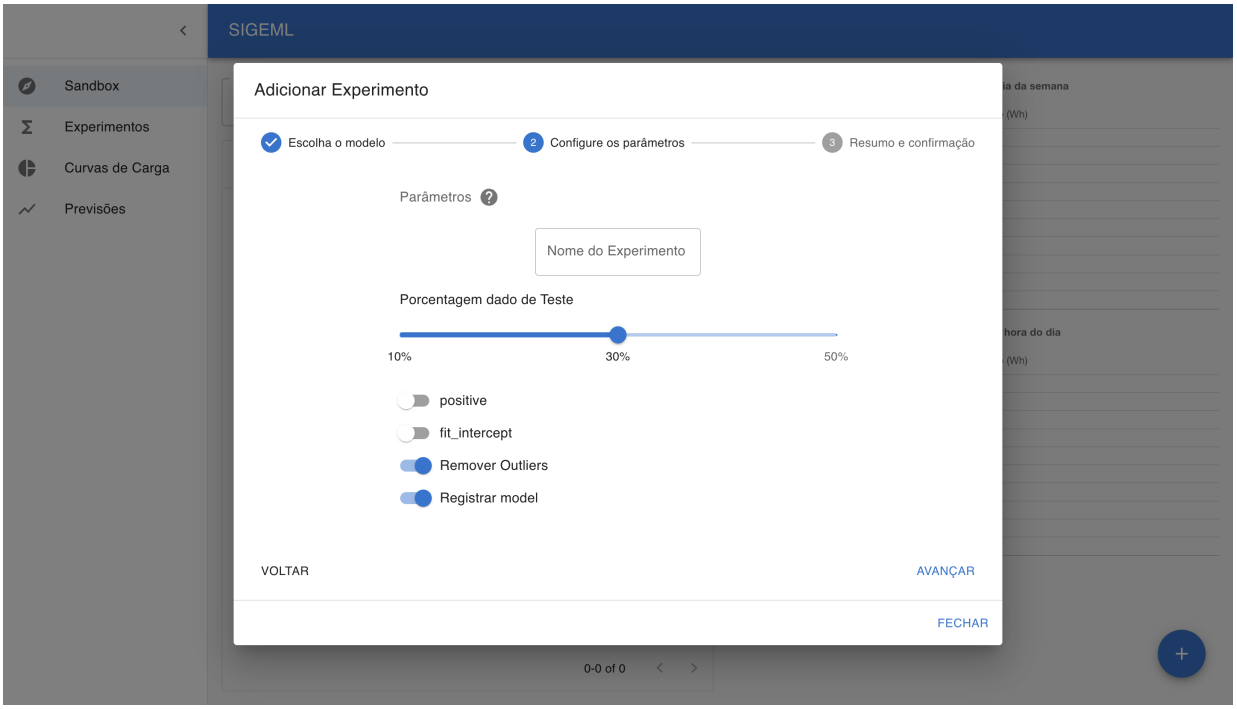


Figura 35 – Tela do *Sandbox*: Formulário de seleção dos parâmetros



Como esta página se concentra apenas em disponibilizar um mecanismo de busca e exibição do histórico de experimentos e seus detalhes, ela não contém um fluxo com início e fim claros. Nesse sentido, considerando uma listagem paginada de experimentos em ordem cronológica de criação, exibida na Figura 38, o usuário tem a opção de aplicar

Figura 36 – Tela do *Sandbox*: Formulário de confirmação

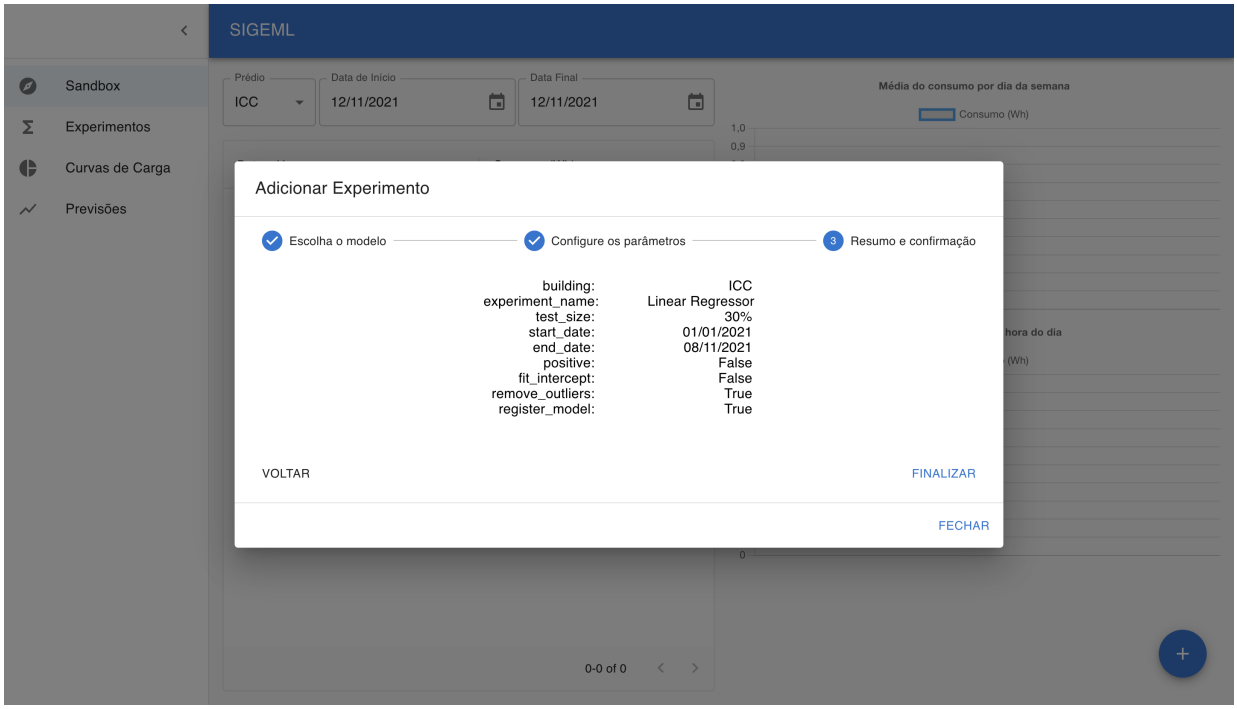
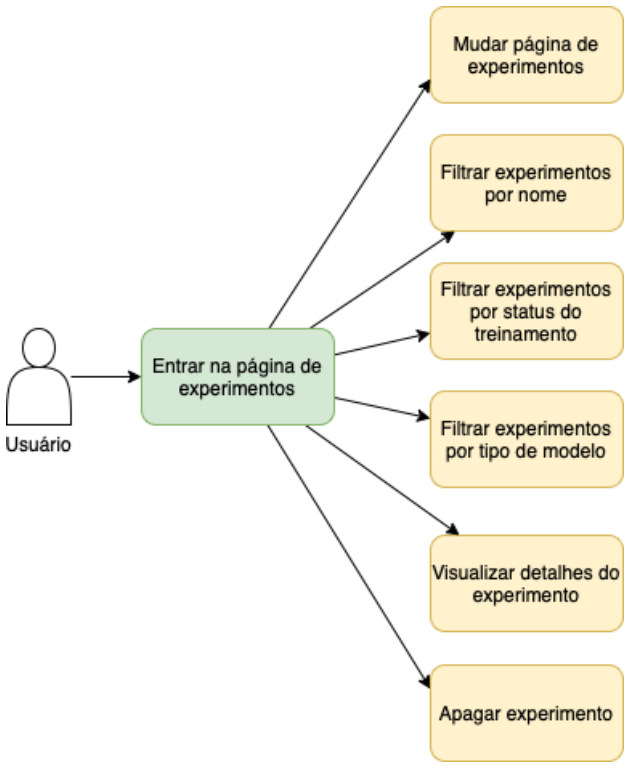


Figura 37 – Diagrama de Fluxo da página de Experimentos



três tipos de filtro: por nome, por status e/ou por modelo utilizado, como mostra a Figura 39.

Além disso, para cada experimento, há a possibilidade de verificar qual o resultado

Figura 38 – Tela com lista de experimentos.

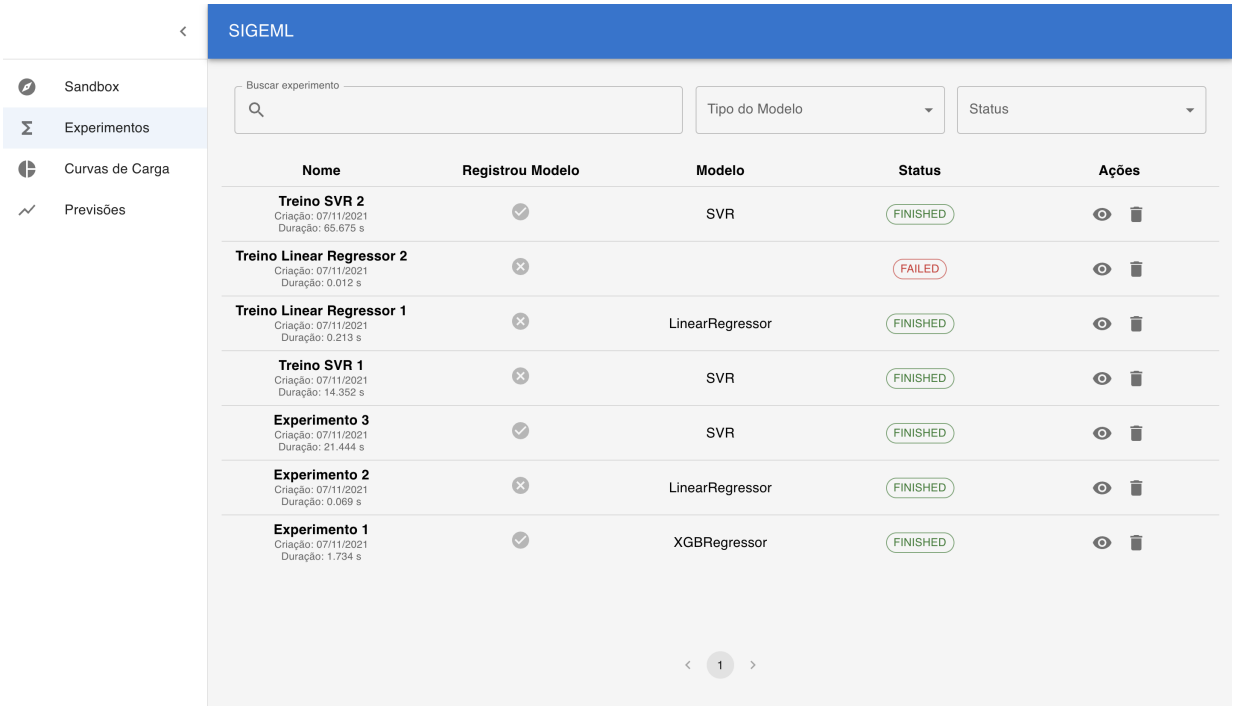
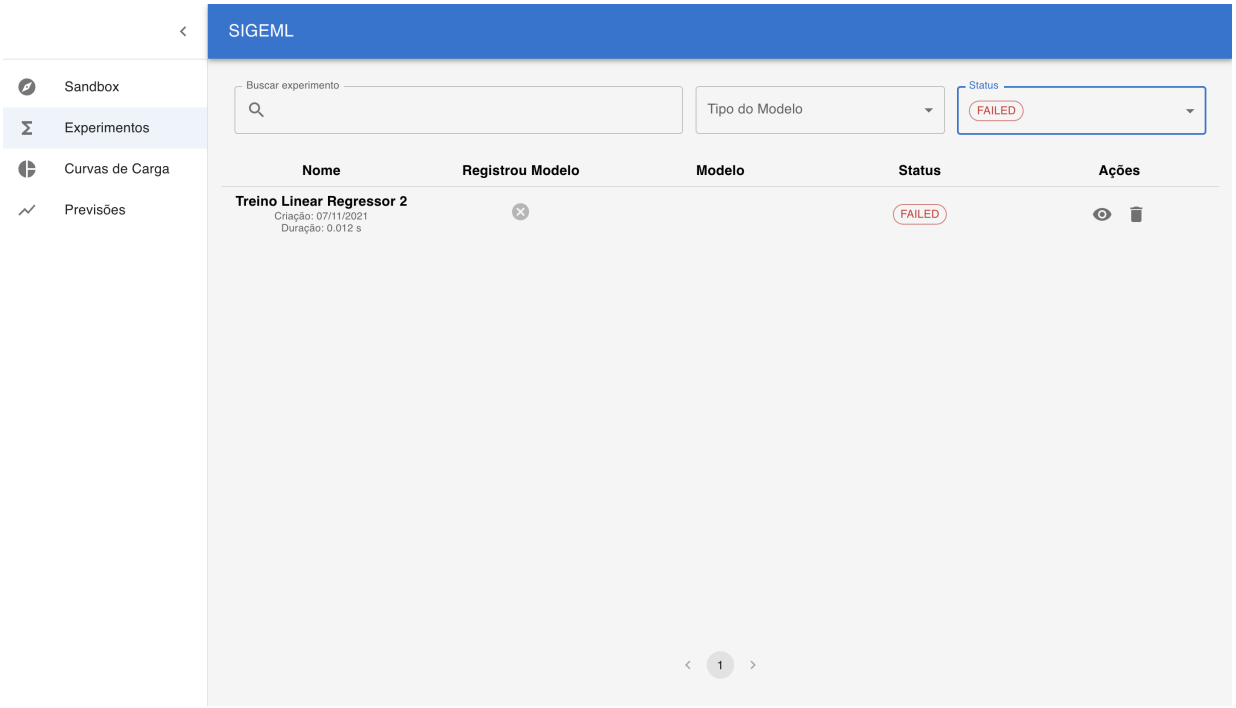
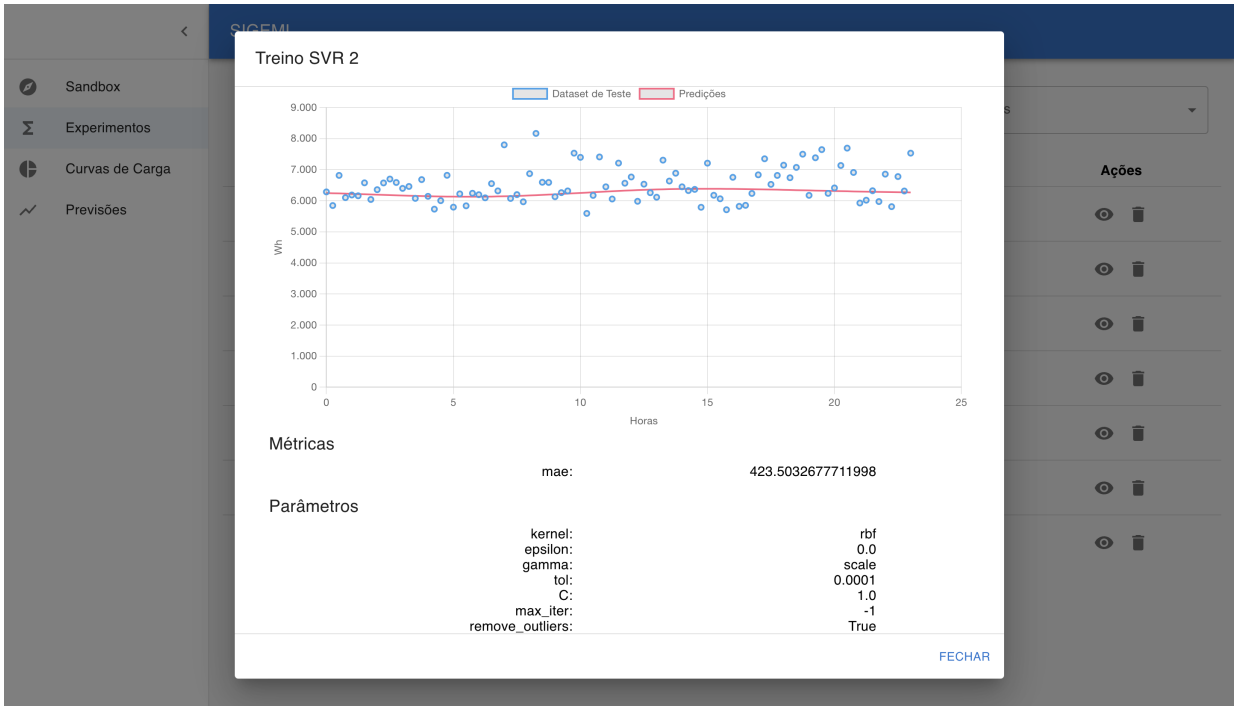


Figura 39 – Tela com filtragem de experimentos.



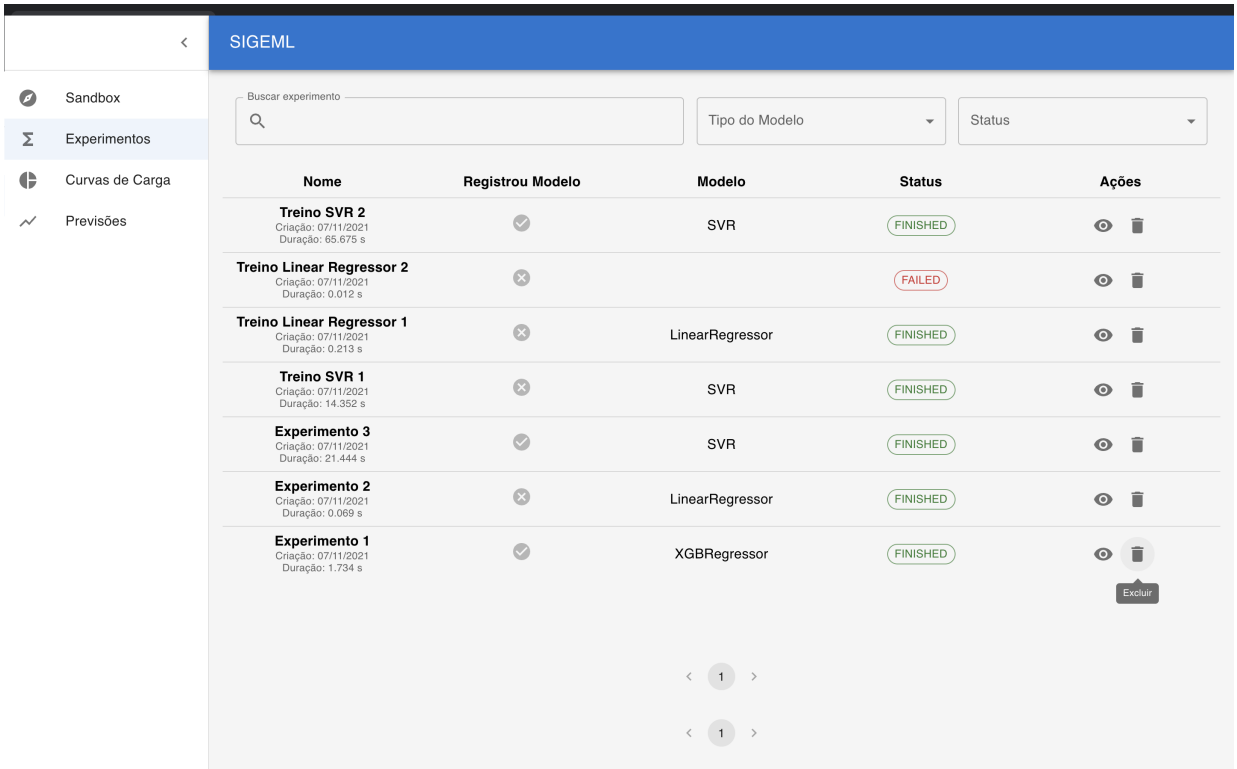
alcançado com sua execução, ou seja, a curva de carga gerada em relação ao dado de teste utilizado, assim como a métrica obtida. Nesse mesmo detalhamento, é possível ver os parâmetros do modelo e o período de tempo para o dado utilizados no treinamento, como exemplifica a Figura 40.

Figura 40 – Tela com detalhes de um experimento.



Por último, também é permitido a exclusão de um determinado experimento por meio do clique no ícone de deleção, mostrado em mais detalhe na Figura 41.

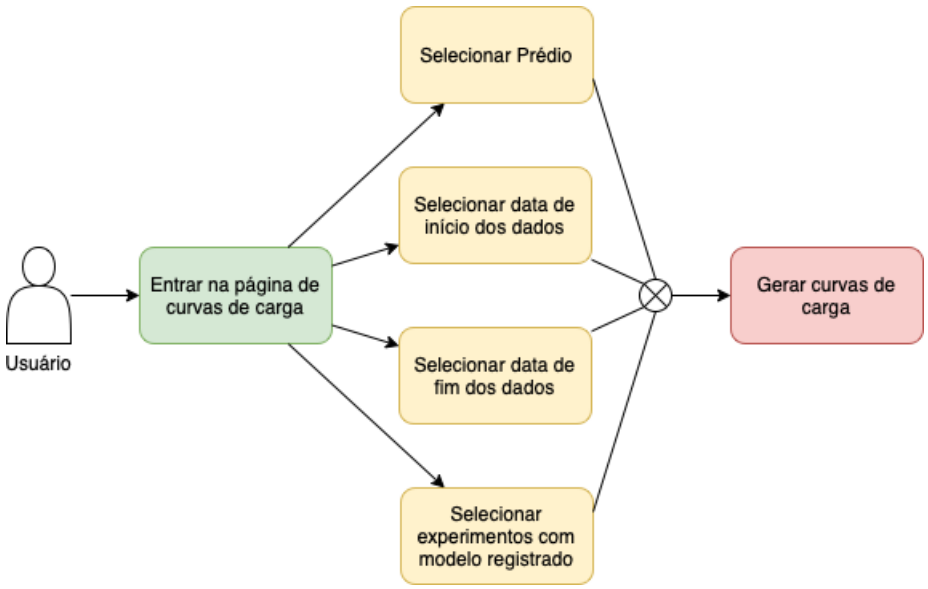
Figura 41 – Destaque do ícone de apagar experimento.



5.3.2.3 Curvas de Carga

A página de curvas de carga permite ao usuário a comparação entre experimentos previamente criados. O diagrama da Figura 42 ilustra as etapas necessárias para realizar tal comparação.

Figura 42 – Diagrama de Fluxo da página de Curvas de Carga



Dessa maneira, ao entrar na página de curvas de carga, o usuário poderá, em qualquer ordem, selecionar o prédio, a data de início e fim para filtragem dos dados provenientes do SIGE e por fim os experimentos que deseja comparar. As curvas de carga com as configurações selecionadas são automaticamente geradas pelo sistema e ilustradas através de um gráfico, como exibido na Figura 43.

5.3.2.4 Predições em tempo real

A página de predições em tempo real permite a visualização das predições feitas pela LSTM mantida no banco de dados do SIGEML para um determinado prédio previamente selecionado pelo usuário. A Figura 44 ilustra o fluxo de etapas necessárias para construção do gráfico em tempo real da página.

A Figura 45 exibe um gráfico construído a partir de da seleção de um prédio por parte do usuário.

5.4 Resultados

Considerando os objetivos mostrados na Seção 1.2.1, os requisitos propostos para o SIGEML apresentados nesta Seção 5 e a ferramenta desenvolvida e disponibilizada ao fim deste trabalho, pode-se afirmar que obtivemos êxito em sua execução e entrega.

Figura 43 – Tela das Curvas de Carga.

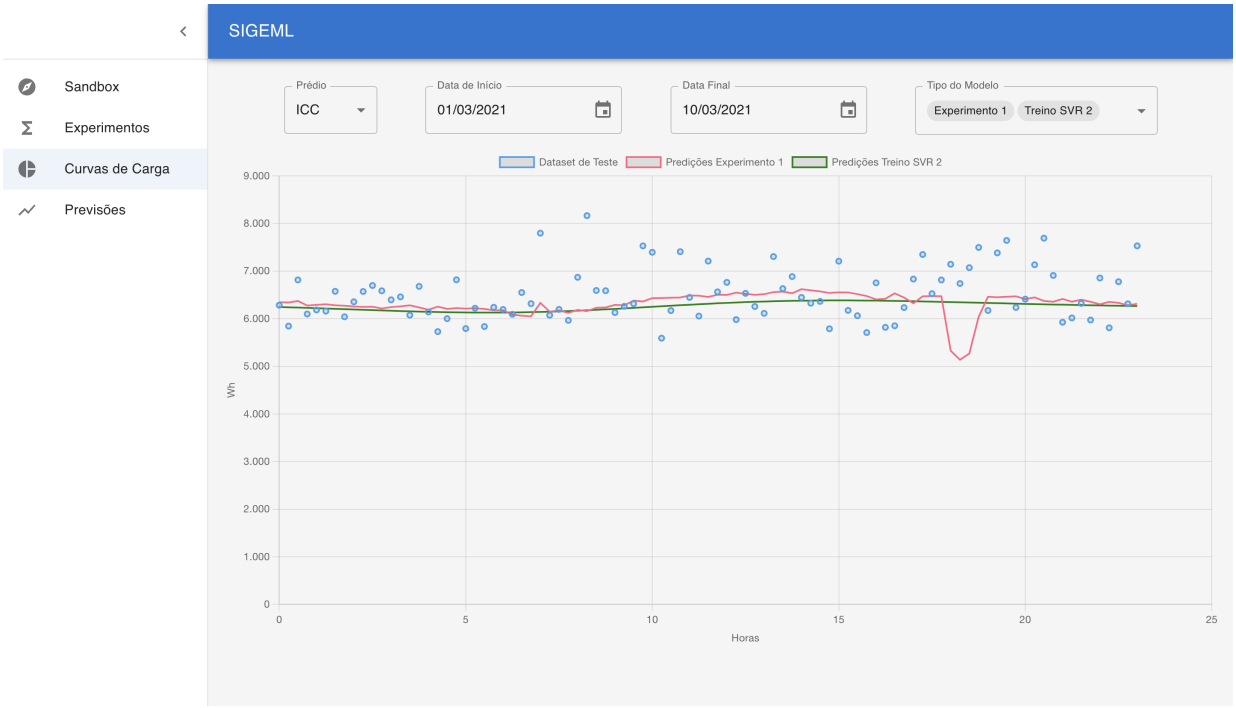


Figura 44 – Diagrama de Fluxo da página de Predições em tempo real

Tal sistema, que auxilia na experimentação e criação de curvas de carga e na predição de séries temporais de consumo, servirá como uma extensão do SIGE, provendo funcionalidades úteis para possíveis funcionários responsáveis e interessados pela gestão energética da UnB.

Apesar disso, ao longo do desenvolvimento do SIGEML, foram encontradas algumas dificuldades, tais como:

- Problemas na coleta de medições de consumo via API do SIGE;
- Baixa amostragem de dados para treinamento das LSTM's;
- Necessidade de um arquitetura complexa para atendimento dos requisitos e objetivos;
- Pouco tempo de desenvolvimento disponível, resultando em:
  - Ausência de testes automáticos de integração;

Figura 45 – Tela com predições em tempo real.

  - Impossibilidade de realizar mais experimentos com outros modelos de curva de carga;

- Integração de gráfico em tempo real no *Front-end* consumindo dados do Broker MQTT.

Esses contratempos, no entanto, não impossibilitaram por completo a construção do SIGEML, abrindo a possibilidade para possíveis trabalhos futuros abrangendo esses pontos. Outros itens tidos como melhorias no sistema serão abordados na Seção **??**.

6 Considerações finais

Este capítulo trata das considerações finais e conclusões alcançadas neste trabalho, bem como dos possíveis trabalhos futuros.

Na Seção 6.1, é feito um resumo sobre o sistema SIGEML e como ele atende os objetivos almejados neste trabalho. Já a Seção 6.2 discorre sobre os possíveis pontos de melhoria do SIGEML que podem servir de trabalhos futuros.

6.1 Conclusão

Neste trabalho, foi desenvolvido o SIGEML, um sistema que provê uma API em conjunto com interface gráfica intuitiva, possibilitando a criação e consulta de curvas de cargas a partir de modelos matemáticos e também a previsão periódica do consumo energético dos prédios da UnB.

O *front-end*, criado com a biblioteca *ReactJS*, juntamente com *Redux*, uma ferramenta de gerenciamento de estado, e *Material UI*, de componentes customizáveis *React*, propicia ao usuário uma interface intuitiva e com boa usabilidade. Nela, há a exibição de gráficos, a listagem do histórico de experimentos e formulários acessíveis, que permitem testes com diferentes modelos de *machine learning*.

Já o *back-end*, implementado na linguagem de programação *Python*, utiliza das bibliotecas *mlflow* e *FastAPI* para disponibilizar uma API independente para a gestão do ciclo de vida de modelos de *machine learning* responsáveis pela geração de curvas de carga. Nele, também há um componente que gera predições de séries temporais de consumo energético em tempo real, enviando-as via protocolo MQTT para um *broker*.

Dessa forma, tendo em vista os requisitos descritos em 5, considera-se essa arquitetura como satisfatória, pois, por meio dela, são disponibilizados dois tipos distintos de acesso ao sistema: um via interface gráfica e outro via API, o que permite o uso do SIGEML tanto por usuários quanto por outros serviços ou sistemas.

Além disso, vale destacar o trabalho inicial realizados antes do desenvolvimento do sistema em si, constituído pela pesquisa bibliográfica seguida da realização de experimentos, como mostrado no Capítulo 4. Tal fase auxiliou no entendimento e na escolha de modelos a serem usados futuramente, representando uma evidência inicial para a definição da viabilidade do SIGEML como um todo.

6.2 Trabalhos Futuros

Tendo em vista as dívidas técnicas compostas dos requisitos não cumpridos neste trabalho, assim como melhorias conhecidas para o sistema, os seguintes itens podem servir de proposta para possíveis trabalhos futuros:

- Requisitos não cumpridos de acordo com as Tabelas 6 e 7:
  - R12: filtrar experimentos por data de criação;
  - R15: selecionar o número de períodos retornados na predição em tempo real;
  - R17: exportar modelos salvos como arquivos *.json* ou *.pickle*;
  - R18: gerenciamento de usuários e permissões;
  - R19: ordenar experimentos por nome;
  - R20: ordenar experimentos por data de criação;
  - R21: permitir o treinamento de LSTM's com parâmetros configuráveis pelo usuário;
- Utilizar outros dados como temperatura e umidade no treinamento das LSTM's;
- Permitir o *upload* de arquivos *.csv* ou *.xlsx* para a criação de experimentos;
- Adicionar mais modelos de curvas de carga;
- Permitir seleção de hora nos campos de data de início de fim;
- Permitir seleção de algoritmo de remoção de *outliers*;
- Permitir aplicação de normalização no dado de treino;
- Implementar Integração Contínua e Entrega Contínua;
- Fazer *deploy* do sistema.

Referências

ANGARITA, J. A. C. et al. *Ações de Eficiência Energética Associadas à Geração Distribuída*. [S.l.: s.n.], 2020. Citado 2 vezes nas páginas 23 e 29.

Ariyo, A. A.; Adewumi, A. O.; Ayo, C. K. Stock price prediction using the arima model. In: *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*. [S.l.: s.n.], 2014. p. 106–112. Citado na página 42.

AWAD, M. *Efficient learning machines : theories, concepts, and applications for engineers and system Designers*. New York: Apress Open, 2015. ISBN 9781430259909. Citado 2 vezes nas páginas 35 e 36.

BARTLETT, P. *Introduction to Time Series Analysis. Lecture 11.* 2010. Citado na página 45.

BIANCHI, F. M. et al. Recurrent neural networks for short-term load forecasting. *SpringerBriefs in Computer Science*, Springer International Publishing, 2017. ISSN 2191-5776. Disponível em: <http://dx.doi.org/10.1007/978-3-319-70338-1>. Citado 4 vezes nas páginas 49, 50, 51 e 52.

CHATTERJEE, S.; SIMONOFF, J. *Handbook of Regression Analysis*. [S.l.]: John Wiley & Sons, Ltd, 2012. i-xiv p. ISBN 9781118532843. Citado 3 vezes nas páginas 30, 31 e 32.

CHRISTENSEN, J. *Nonlinear Optimization of Vehicle Safety Structures : Modeling of Structures Subjected to Large Deformations*. Saint Louis, MO: Elsevier Science, 2015. ISBN 978-0-12-804424-7. Citado 2 vezes nas páginas 37 e 38.

Contreras, J. et al. Arima models to predict next-day electricity prices. *IEEE Transactions on Power Systems*, v. 18, n. 3, p. 1014–1020, 2003. Citado na página 42.

DEISENROTH, M. P.; FAISAL, A. A.; ONG, C. S. *Mathematics for Machine Learning*. [S.l.]: Cambridge University Press, 2020. Citado 3 vezes nas páginas 30, 32 e 33.

ECLIPSE. *An open source MQTT broker*. 2021. Disponível em: <https://mosquitto.org>. Citado na página 58.

ENGELSDORFF, T. S. *Métodos em machine learning para construção de curvas de carga a partir de medições*. 2019. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Energia) — Universidade de Brasília, Brasil. Citado 2 vezes nas páginas 27 e 28.

FLICK, D. et al. Machine learning based analysis of factory energy load curves with focus on transition times for anomaly detection. *Procedia CIRP*, v. 93, p. 461–466, 2020. ISSN 2212-8271. 53rd CIRP Conference on Manufacturing Systems 2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2212827120306855>. Citado na página 28.

FRANCISQUINI, A. A. *Estimação de Curvas de Carga em Pontos de Consumo e em Transformadores de Distribuição*. 2006. Dissertação (Mestrado em Engenharia Elétrica)

— Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista "Júlio de Mesquita Filho", Ilha Solteira, Brasil. Citado 2 vezes nas páginas 27 e 28.

GRAVES, A. *Sequence Transduction with Recurrent Neural Networks*. 2012. Citado na página 49.

GREGOR, K. et al. *DRAW: A Recurrent Neural Network For Image Generation*. 2015. Citado na página 49.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc., 2001. (Springer Series in Statistics). Citado na página 39.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, v. 9, p. 1735–80, 12 1997. Citado na página 52.

HSU, C.-W.; CHANG, C.-C.; LIN, C.-J. *A Practical Guide to Support Vector Classification*. [S.l.], 2003. Disponível em: <http://www.csie.ntu.edu.tw/~cjlin/papers.html>. Citado na página 35.

HUDAIB, A. et al. Requirements prioritization techniques comparison. Canadian Center of Science and Education, v. 12, n. 2, p. 62, jan 2018. Disponível em: <https://doi.org/10.5539%2Fmas.v12n2p62>. Citado na página 77.

HYNDMAN, R.; ATHANASOPOULOS, G. *Forecasting: principles and practice*. OTexts, 2018. ISBN 9780987507112. Disponível em: <https://otexts.com/fpp2>. Citado 9 vezes nas páginas 39, 40, 41, 42, 43, 44, 46, 48 e 66.

HYNDMAN, R. J. *Forecasting with long seasonal periods*. 2010. <https://robjhyndman.com/hyndsight/longseasonality/>. Citado na página 68.

JUPYTER. *Jupyter Library Home Page*. 2021. Disponível em: <https://jupyter.org/index.html>. Citado na página 58.

KAVASSERI, R. G.; SEETHARAMAN, K. Day-ahead wind speed forecasting using f-arima models. *Renewable Energy*, v. 34, n. 5, p. 1388–1393, 2009. ISSN 0960-1481. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0960148108003327>. Citado na página 42.

KERAS. *Layer weight initializers*. 2021. Disponível em: <https://keras.io/api/layers/initializers/>. Citado na página 58.

KERAS. *Layer weight initializers*. 2021. Disponível em: <https://keras.io/api/layers/initializers/>. Citado na página 68.

KONSTANTINOV, A. V.; UTKIN, L. V. Interpretable machine learning with an ensemble of gradient boosting machines. *Knowledge-Based Systems*, v. 222, p. 106993, 2021. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705121002562>. Citado 2 vezes nas páginas 37 e 38.

LIU, H.; SHAH, S.; JIANG, W. On-line outlier detection and data cleaning. *Computers & Chemical Engineering*, Elsevier BV, v. 28, n. 9, p. 1635–1647, ago. 2004. Disponível em: <https://doi.org/10.1016/j.compchemeng.2004.01.009>. Citado na página 42.

MATHWORKS. *Hampel Filter*. 2021. <https://www.mathworks.com/help/dsp/ref/hampelfilter.html>. Accessed: 2021-05-12. Citado na página 42.

MATPLOTLIB. *Matplotlib: Visualization with Python*. 2021. Disponível em: <https://matplotlib.org/>. Citado na página 58.

MLFLOW. *Mlflow documentation*. 2021. Disponível em: <https://mlflow.org/docs/latest/index.html>. Citado 2 vezes nas páginas 58 e 83.

NCERT. *Statistics for economics : Textbook for class XI*. New Delhi: National Council of Educational Research and Training, 2018. ISBN 81-7450-497-4. Citado na página 63.

OORD, A. van den et al. *WaveNet: A Generative Model for Raw Audio*. 2016. Citado na página 49.

PANDAS. *Pandas Documentation*. 2021. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/index.html#>. Citado na página 59.

POSTGRESQL. *The World's Most Advanced Open Source Relational Database*. 2021. Disponível em: <https://www.postgresql.org>. Citado na página 59.

PYTHON. *Python 3.9.5 documentation*. 2021. Disponível em: <https://docs.python.org/3/>. Citado 2 vezes nas páginas 59 e 82.

RAMíREZ, S. *FastAPI framework, high performance, easy to learn, fast to code, ready for production*. 2021. Disponível em: <https://fastapi.tiangolo.com/>. Citado na página 58.

RASCHKA, S. *Python machine learning : machine learning and deep learning with python, scikit-learn, and tensorflow 2*. Birmingham: Packt Publishing, Limited, 2019. ISBN 1-78995-829-6. Citado na página 38.

REACTJS. *React: A JavaScript library for building user interfaces*. 2021. Disponível em: <https://reactjs.org/>. Citado 2 vezes nas páginas 59 e 86.

REDIS. *In-memory data structure store, used as a database, cache, and message broker*. 2021. Disponível em: <https://redis.io>. Citado na página 59.

SAGI, O.; ROKACH, L. Explainable decision forest: Transforming a decision forest into an interpretable tree. *Information Fusion*, v. 61, p. 124–138, 2020. ISSN 1566-2535. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1566253519307869>. Citado na página 37.

SCIKITLEARN. *MinMaxScaler*. 2021. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>. Citado 2 vezes nas páginas 68 e 70.

SCIKITLEARN. *scikit-learn Machine Learning in Python*. 2021. Disponível em: <https://scikit-learn.org/stable/>. Citado na página 59.

SCIKITLEARN. *sklearn.linear_model.LinearRegression*. 2021. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression>. Citado na página 85.

SCIKITLEARN. *sklearn.svm.SVR*. 2021. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html?highlight=svr#sklearn.svm.SVR>. Citado na página 85.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014. ISBN 1107057132. Citado na página 30.

SHUMWAY, R.; STOFFER, D. *Time Series Analysis and Its Applications: With R Examples*. [S.l.]: Springer International Publishing, 2017. (Springer Texts in Statistics). ISBN 9783319524528. Citado na página 39.

STATSMODELS. *Statistical models, hypothesis tests, and data exploration*. 2021. Disponível em: <https://www.statsmodels.org/stable/index.html>. Citado na página 59.

STEWART, J. *Multivariable calculus*. Belmont, CA: Brooks/Cole Cengage Learning, 2012. ISBN 0-538-49787-4. Citado na página 34.

SUTHERLAND, J.; SCHWABER, K. *The Scrum Guide*. 2021. Disponível em: <https://scrumguides.org/scrum-guide.html>. Citado na página 79.

SVOZIL, D.; KVASNICKA, V.; POSPICHAL, J. Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, v. 39, n. 1, p. 43–62, 1997. ISSN 0169-7439. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169743997000610>. Citado 2 vezes nas páginas 47 e 49.

TENSORFLOW. *An end-to-end open source machine learning platform*. 2021. Disponível em: <https://www.tensorflow.org/>. Citado na página 59.

TENSORFLOW. *Tensorflow Data Windowing*. 2021. Disponível em: <https://www.tensorflow.org/tutorials/structured_data/time_series#data_windowing>. Citado 2 vezes nas páginas 50 e 51.

TSAY, R. S. *Analysis of financial time series*. 3. ed.. ed. [S.l.]: Wiley-Interscience, 2010. (Wiley series in probability and statistics). ISBN 978-0-470-41435-4. Citado na página 42.

WINSTON, P. *6.034 Artificial Intelligence*. Massachusetts Institute of Technology: MIT, 2010. Disponível em: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/#>. Citado na página 33.

XGBOOST. *Introduction to Boosted Trees*. 2021. Disponível em: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>. Citado na página 39.

XGBOOST. *xgboost.XGBRegressor*. 2021. Disponível em: <https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBRegressor>. Citado 2 vezes nas páginas 59 e 86.