UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANTÔNIO CORREIA DE SÁ BARRETO NETO

Modelos de Machine Learning baseados no padrão de uso do usuário para estimar consumo energético e utilização de dispositivos do smartphone

Recife

2022

ANTÔNIO CORREIA DE SÁ BARRETO NETO

Modelos de Machine Learning baseados no padrão de uso do usuário para estimar consumo energético e utilização de dispositivos do smartphone

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Área de Concentração: Redes de Computadores e Sistemas Distribuídos

Orientador: Paulo Romero Martins Maciel

Recife

2022

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

B273m Barreto Neto, Antônio Correia de Sá
Modelos de Machine Learning baseados no padrão de uso do usuário para estimar consumo energético e utilização de dispositivos do smartphone / Antônio Correia de Sá Barreto Neto. – 2022.
159 f.:il., fig, tab.

Orientador: Paulo Romero Martins Maciel.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022.

Inclui referências e apêndices.

1. Redes de computadores. 2. Aprendizagem de máquina. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004.6 CDD (23. ed.) UFPE - CCEN 2022-196

Antônio Correia de Sá Barreto Neto

“Modelos de Machine Learning baseados no padrão de uso do usuário para estimar consumo energético e utilização de dispositivos do smartphone”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Aprovado em: 16/09/2022.

_____

Orientador: Prof. Dr. Paulo Romero Martins Maciel

BANCA EXAMINADORA

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Dr. Germano Crispim Vasconcelos
Centro de Informática / UFPE

_____

Prof. Dr. Eduardo Antonio Guimarães Tavares
Centro de Informática / UFPE

_____

Prof. Dr. Jose Neuman de Souza
Departamento de Computação / UFC

_____

Profa. Dra. Patricia Takako Endo
Universidade de Pernambuco / Campus Caruaru

Dedico este trabalho a todos os meus amigos, a meus pais e a meu irmão, pois sem a ajuda deles eu não poderia ter chegado até aqui.

AGRADECIMENTOS

Agradeço, primeiramente, a meus pais, Osvaldo Ribeiro e Rosário Sá Barreto (*in memoriam*), que sempre acreditaram no meu pontencial e nele investiram, apoiando-me em todas as etapas de minha vida, de tal forma que hoje foi possível chegar até aqui. Agradeço à minha mãe, Rosário Sá Barreto (*in memoriam*), por ter me ensinado o valor do estudo e me possibilitado enxergar o quanto a educação é transformadora.

Agradeço ao professor Paulo Maciel, por ter acreditado no potencial de minha pesquisa, me orientando mesmo antes de possuir vínculo oficial com o programa de doutorado, sendo não apenas orientador, mas parceiro de pesquisa.

Agradeço à minha grande amiga, Príscila Alves Lima, que me apresentou ao professor Paulo Maciel e, nos momentos difíceis da pesquisa, esteve ao meu lado, dando-me forças para perseverar.

Agradeço ao meu amigo Marco Aurélio Tomaz Mialaret Júnior, que, desde o mestrado, sempre enxergou grande potencial em minha pesquisa, motivando-me a continuar minhas investigações.

Agradeço a meu amigo Felipe Costa Farias, que agiu, em meu trabalho, como um verdadeiro crítico para o desenvolvimento das teorias de inteligência artificial.

Agradeço a meu amigo Bruno Cartaxo, que, desde os tempos de faculdade, sempre trabalhou em parceria comigo, apoiando-me nos meus desafios profissionais e auxiliando-me nas correções dos artigos da pesquisa.

Agradeço à companhia de meus cachorros, Che Guevara (*in memoriam*) e Snoopy (*in memoriam*), durante 17 anos e 6 meses e 16 anos e 3 meses, respectivamente. Foram eles grandes companheiros meus durante a graduação, o mestrado e o doutorado, dando-me forças para superar a morte de minha mãe, no início do doutorado. Eles me chamavam para passear sempre que eu estava bastante estressado com os desdobramentos da pesquisa. Esforçaram-se bastante, mas eu não consegui dar esse agradecimento a eles em vida.

Agradeço, por fim, a todos que contribuíram com o processo da pesquisa, mesmo de forma indireta, pois sei que sem esse apoio a realização deste trabalho ficaria mais difícil.

"Experimental observations are only experience carefully planned in advance, and designed to form a secure basis of new knowledge."(FISHER,1935,p.8)

RESUMO

O crescente uso de smartphones nas tarefas cotidianas tem motivado muitos estudos sobre a caracterização do consumo de energia para melhorar a eficiência energética dos dispositivos componentes dos smartphones e aumentar o tempo de uso do usuário. Em 2013, o consórcio de empresas responsável pelo desenvolvimento do Android disponibilizou no Android 4.4, uma ferramenta responsável por calcular o consumo instantâneo do dispositivo e, a partir dessa informação, realizar predições de tempo remanescente de uso. No entanto, após uma análise mais profunda, é possível perceber que essas ferramentas desenvolvidas e já presentes nas versões mais recentes do Android são mais adequadas para realizar o estudo a respeito do consumo energético do smartphone no passado, sendo muito suscetíveis a alterações significativas, a depender do comportamento do usuário. Baseados nessas constatações, realizamos uma pesquisa que visa extrair, a partir do monitoramento do smartphone, informações que caracterizem o padrão de comportamento do usuário independente do modelo do smartphone e conceber modelos de aprendizagem de máquina que permitam relacionar o uso dos dispositivos que compõem o smartphone ao padrão de comportamento do usuário. Para atingir esse objetivo, durante a realização dessa pesquisa, propomos um *workflow* para extração do padrão de comportamento do usuário e construção de modelos de aprendizagem de máquina que relacionem esse padrão de comportamento ao uso dos dispositivos que compõem o smartphone utilizando modelos de aprendizagem de máquina rasos (*shallow*), em oposição às redes neurais profundas (*deep*), utilizando módulos de software construídos ao longo dessa pesquisa. Foi possível perceber, através da realização dos experimentos realizados com os módulos de software propostos, que utilizamos vários regressores e os resultados são relevantes e consistentes em termos do erro alcançado. Com algoritmos de aprendizagem de máquina simples e rápidos de serem ajustados, treinados e testados, conseguimos modelar uso dos dispositivos que compõem o smartphone baseados no padrão de comportamento do usuário.

Palavras-chaves: consumo de energia; uso de dispositivos; aprendizagem de máquina; inteligência artificial; padrão de comportamento do usuário.

ABSTRACT

The increasing usage of smartphone in daily activities has motivated many studies on energy consumption characterization to improve smartphone devices' effectiveness and increase user usage time. In 2013, the consortium of companies responsible for the development of Android made available in Android 4.4, a tool responsible for calculating the instantaneous consumption of the device and make predictions of the remaining time of use. However, after a deep analysis, it is possible to perceive that these tools developed and already present in the newer versions of Android are more suitable for carrying out the study regarding the smartphone energy consumption in the past, being very susceptible to significant changes depending on user behavior. Based on these findings, we performed a research that aims to extract, from smartphone monitoring, information that characterizes the user's behavior pattern regardless of the smartphone model and to conceive machine learning models that allow relating the use of devices that make up the smartphone to the user's behavior pattern. To achieve this goal, during this research, we propose a workflow for extracting the pattern of user behavior and construction of machine learning models that relate this behavior pattern to the use of devices that make up the smartphone using shallow machine learning models, as opposed to deep neural networks, using software modules built throughout this research. Through the experiments with the proposed software modules, we could see that we used several regressors, and the results are relevant and consistent in terms of the achieved error. With simple and quick machine learning algorithms to be adjusted, trained, and tested, we could model the use of the devices that make up the smartphone based on the user's behavior pattern.

Keywords: energy consumption; devices usage. machine learning; artificial intelligence; user usage behavior pattern.

LISTA DE FIGURAS

LISTA DE CÓDIGOS

LISTA DE MÓDULOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
APk	Application Package
ARM	Advanced RISC Machine
ConvLSTM	Convolutional Long Short-Term Memory
CPU	Central Processing Unit
CSV	Comma Separated Values
DVFS	Dynamic Voltage Frequency Scaling
GA	Genetic Algorithm
GPU	Graphical Processing Unit
IC	Intervalo de Confiança
IMEI	International Mobile Equipment Identity
Kb	Kilobits
LGBM	Light Gradient Boosting Machine
LSTM	Long Short Term Memory
mA	miliAmpère
mAh	miliAmpère-hora
MSE	Mean Squared Error
mW	miliWatts
OHA	Open Handset Alliance
OMP	Orthogonal Matching Pursuit
PCA	Principal Component Analysis
REST	Representational State Transfer
RFE	Recursive Feature Elimination
RMSE	Root Mean Squared Error
SDR	Standard Deviation Reduction

VFS Voltage Frequency Scaling

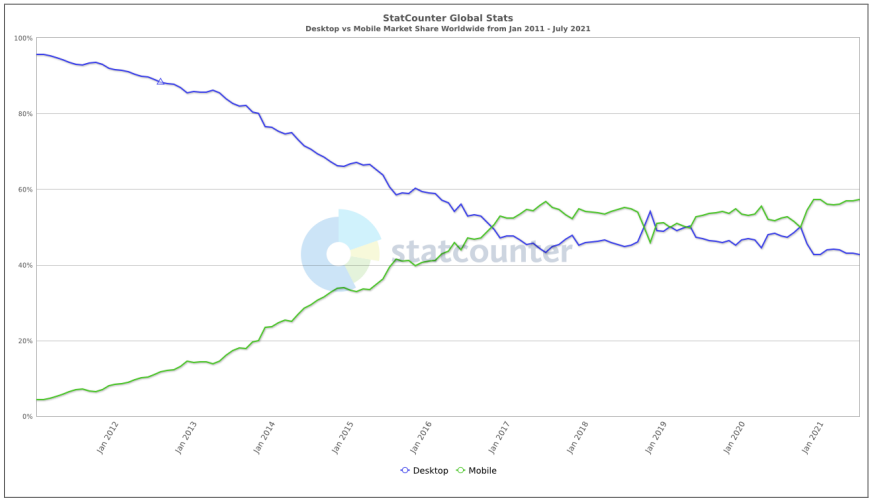
WSGI Web Server Gateway Interface

SUMÁRIO

1 INTRODUÇÃO

A participação de dispositivos móveis no mercado aumentou nos últimos dez anos (STATS, 2021). Em 2011, a participação de mercado de computadores desktop foi de 91,96% vs. 8,04% dos smartphones, e, em julho de 2021, a participação de mercado dos dispositivos móveis aumentou para 56,9% vs. 43,1% do desktop, conforme mostrado na Figura 1.

Figura 1 – Desktop vs Mobile Market Share em todo o mundo - Julho de 2021



Fonte: STATS (2021)

De acordo com STATISTA(2021), o número de assinaturas de smartphones em todo o mundo aumentou de 3,668 bilhões de pessoas em 2016 para 6,378 bilhões de pessoas em 2021, representando um aumento de 173%.

Essa tendência, no entanto, traz seus desafios. Um dos mais proeminentes é o alto consumo de energia dos smartphones, diminuindo a disponibilidade deles devido à limitada vida útil da bateria (CHEN; SEN, 2016) (LI et al., 2018). Em 2005, PARADISO; STARNER(2005) desenvolveram um estudo que demonstrou que a evolução da tecnologia de baterias não está no mesmo ritmo dos outros dispositivos de smartphones.

Em 2018, a AUTHORITY(2021) realizou uma pesquisa para avaliar se a capacidade da bateria evoluiu. Eles concluíram que, em cinco anos, a capacidade da bateria passou de 3300miliAmpère-hora (mAh) para cerca de 3500mAh e estimaram que, em 2020, as baterias com 4000mAh deveriam ser padrão, o que foi confirmado pelo uso de baterias com essa capacidade em smartphones como Samsung Galaxy S20 (ARENA, 2021a) e Samsung Galaxy S21 (ARENA, 2021b).

Com base nesses achados, diversos tipos de pesquisas têm sido realizados para otimizar o uso dos dispositivos do smartphone e, consequentemente, seu consumo de energia. Em 2019, PRAMANIK et al.(2019) realizaram uma revisão do estado da arte das baterias de smartphones e do uso de energia e concluíram que há uma limitação da capacidade da bateria, a qual é determinada por suas propriedades químicas, não podendo ser aumentada além de um limite específico. Os autores sugeriram usar a energia com cautela. Eles disseram que é necessária uma compreensão rigorosa da necessidade de energia de cada componente e do consumo de um smartphone.

Assim, com o intuito de otimizar o consumo de energia dos dispositivos móveis, alguns autores propuseram estratégias baseadas em aprendizagem por reforço para ajustar a frequência da CPU. SHAFIK et al.(2015) propõem uma abordagem online capaz de minimizar a energia através da adaptação. O núcleo dessa abordagem é um algoritmo de aprendizagem por reforço que seleciona adequadamente a Voltage Frequency Scaling (VFS) apropriada com base nas previsões de carga de trabalho para atender aos requisitos de desempenho dos aplicativos. A abordagem proposta é implementada como um regulador de energia no Linux e amplamente validada em um Advanced RISC Machine (ARM) Cortex-A8, rodando diferentes aplicações de referência e minimizando o consumo de energia em até 33% em comparação com abordagens de escalonamento da CPU *predictive* e baseada em conhecimento. Escalando a abordagem para sistemas multicore, os autores também demonstram a possibilidade de minimizar a energia em até 18% com uma redução de duas vezes no tempo de aprendizagem em comparação com uma abordagem que aprende as mudanças de temperatura e carga de trabalho observando o sensor de temperatura.

TIAN et al.(2018) propuseram uma abordagem de gerenciamento de energia colaborativa em vários dispositivos para verificar uma maneira de encontrar rapidamente uma política de gerenciamento eficiente sob a complexidade cada vez maior de hardware e software. O mecanismo de gerenciamento colaborativo de energia compartilha, periodicamente, conhecimento entre vários dispositivos para acelerar o processo de aprendizagem e melhorar a qualidade das políticas aprendidas, integrando o método proposto com Dynamic Voltage Frequency Scaling (DVFS) nos processadores multicore em dispositivos móveis. Resultados experimentais em seis aplicativos do conjunto de aplicativos do *benchmark* COSMIC mostram que o gerenciamento colaborativo de energia pode atingir uma velocidade média 8 vezes maior e 10% de economia de energia em comparação com abordagens baseadas em aprendizagem.

Outra estratégia adotada para fazer otimizações é avaliar os padrões de uso dos usuários.

WANG et al.(2013) propuseram uma metodologia de rastreamento de bateria baseada em perfil. Segundo os autores, eles podem coletar facilmente registros de uso de bateria por meio de um aplicativo de nível de usuário em qualquer dispositivo. Os autores mostraram que, após coletar registros de uso de bateria de mais de 80000 usuários, é possível estimar o gasto energético de uma aplicação com registros de alto nível de uso de bateria.

Em 2015, KIM et al.(2015) propuseram uma estrutura para monitorar cuidadosamente os eventos do sistema iniciados por interações do usuário e identificar a atividade atual do usuário com base em um modelo de atividade online.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

O estudo a respeito de técnicas de modelagem e redução do consumo energético é um tema bastante em voga, haja vista o aumento do poder computacional dos smartphones sem uma evolução no mesmo ritmo das tecnologias empregadas nas baterias. Esses estudos vêm sendo realizados não apenas no mercado de smartphones mas também de laptops com o emprego de tecnologias existentes nos smartphones para o desenvolvimento de processadores mais eficientes, utilizando a tecnologia de processadores ARM heterogêneos denominada *big.LITTLE*.

A presente pesquisa se iniciou em 2012, em um projeto da Samsung que tinha por objetivo reduzir o consumo energético dos smartphones através da aplicação de técnicas de ajuste da frequência de operação da CPU de acordo com a aplicação em uso.

No entanto, ao final da primeira etapa da referida pesquisa, percebemos que o fator que mais impactava no consumo energético dos smartphones não era apenas um dispositivo ou aplicativo específico e sim o usuário, fato esse corroborado pela pesquisas realizadas por SHYE; SCHOLBROCK; MEMIK(2009) e BONETTO et al.(2012).

De posse desses resultados, resolvemos realizar uma nova etapa da pesquisa que culminou nesta tese de doutorado. Com os resultados obtidos ao final da primeira fase desta pesquisa, precisávamos readequar o método de coleta de dados utilizado para não apenas coletar dados de consumo considerando o caráter temporal das potências instantâneas mas, também, coletar dados de contexto de uso.

Em 2013, o consórcio de empresas responsável pelo desenvolvimento do Android, chamado Open Handset Alliance (OHA), disponibilizou, no Android 4.4, uma ferramenta responsável por calcular o consumo instantâneo do dispositivo e, a partir dessa informação, realizar predições

de tempo remanescente de uso.

Para obter a informação a respeito do que causa o consumo da bateria, o Android utiliza um arquivo disponibilizado pelos fabricantes dos smartphones. Esse arquivo foi nomeado como *power_profile.xml* e reside dentro do pacote frameworks-res.apk (PROJECT, 2022b). De acordo com o guia de implantação do Android (PROJECT, 2022b), os fabricantes de smartphones devem fornecer um perfil de energia de cada um dos dispositivos que compõem o smartphone, definindo o valor de consumo atual e o consumo aproximado de bateria causado por cada um dos dispositivos que compõem o smartphone ao longo do tempo.

Além de construir um mecanismo que analisa o consumo da bateria baseando-se no estado atual, a OHA também desenvolveu, a partir do Android Lollipop,em 2014, uma outra ferramenta bastante útil para os desenvolvedores de aplicativos: a *batterystats* (DEVELOPERS, 2022b). Para simplificar a análise da informação produzida pela ferramenta *batterystats*, o Google desenvolveu um programa chamado Battery Historian (GOOGLE, 2017), que mostra graficamente como cada aplicativo se comportou em termos de consumo de energia, informando sobre os dados trafegados pelas redes móveis e Wi-Fi.

No entanto, após uma análise mais profunda, é possível perceber que essas ferramentas desenvolvidas e já presentes nas versões mais recentes do Android são mais adequadas para realizar o estudo a respeito do consumo energético do smartphone no passado, sendo muito suscetíveis a alterações significativas, a depender do comportamento do usuário.

Ao realizarmos estudos a respeito das pesquisas que estão sendo realizadas na área, percebemos que muitas delas têm se concentrado em extrair o padrão de comportamento do usuário quanto ao uso de aplicativos. Outras têm se concentrado em modelar o consumo energético usando duas abordagens principais: 1) do ponto de vista do estado de uso do dispositivo, 2) do uso de técnicas de previsão de séries temporais. Poucas pesquisas têm buscado relacionar o padrão de comportamento do usuário ao uso dos dispositivos que compõem o smartphone e, diante dessa constatação, estabalecemos os objetivos discutidos na próxima seção.

1.2 OBJETIVOS

Com base nos achados dos estudos anteriores, que demonstram a importância de estudar estratégias para aumentar o tempo de uso do smartphone devido à sua aplicabilidade em muitas situações, investigamos abordagens para estender o tempo de uso do smartphone e entender o comportamento do usuário para modelar o uso dos dispositivos do smartphone

e o respectivo consumo de energia para melhorá-lo. Assim sendo, estabelecemos o seguinte objetivo geral de pesquisa:

- Criar um *workflow* capaz de gerar modelos de uso dos dispositivos que compõem os smartphones baseados no padrão de comportamento dos usuários.

Para alcançarmos esse objetivo, propomos um procedimento metodológico de análise do padrão de comportamento dos usuários e modelagem do consumo energético dos dispositivos que compõem o smartphone baseado nesse padrão de comportamento analisado. Para construir esse procedimento metodológico de tal forma que pudéssemos implementar e gerar os resultados para atender ao objetivo geral desta pesquisa estabelecido acima, enumeramos um subconjunto de passos e metas que caracterizam os objetivos específicos a seguir.

1. Extrair, a partir do monitoramento do smartphone, informações que caracterizem o padrão de comportamento do usuário independente do modelo do smartphone.
2. Avaliar o uso dos dispositivos que compõem o smartphone a partir dos dados coletados no monitoramento das atividades do usuário.
3. Conceber modelos de aprendizagem de máquina que permitam relacionar o uso dos dispoitivos que compõem o smartphone ao padrão de comportamento do usuário.
4. Implementar módulos de software que sejam capazes de automatizar o processo de análise dos dados de uso e criação dos modelos de aprendizagem de máquina.
5. Criar e avaliar cenários que mostrem a viabilidade dos modelos de aprendizagem de máquina e dos módulos de software propostos.

1.3 CONTRIBUIÇÕES

Podemos citar, a seguir, como possíveis contribuições da pesquisa relatada nesta tese de doutorado:

- Definição de um *workflow* para extração do padrão de comportamento do usuário e construção de modelos de aprendizagem de máquina que relacionem esse padrão de comportamento ao uso dos dispositivos que compõem o smartphone.

- Criação de um método para automatizar a análise dos dados do padrão de comportamento do usuário e construção dos modelos de aprendizagem de máquina que relacionam esse padrão de comportamento ao uso dos dispositivos que compõem o smartphone.
- Criação de um método para analisar a mudança no comportamento do usuário através de uma técnica chamada *drift* de dados.
- Definição de um método para avaliar o erro cometido por modelos de aprendizagem de máquina construídos a partir de conjuntos de dados enviesados.
- Utilização de modelos regressores de baixa complexidade computacional capazes de modelar a relação existente entre o padrão de comportamento do usuário e a utilização dos dispositivos que compõem o smartphone.
- Utilização dos modelos criados a partir do smartphone sem aumentar significativamente o consumo energético do smartphone com a utilização dos modelos gerados.

1.4 ORGANIZAÇÃO DA TESE

O restante desta tese é organizado como segue.

O Capítulo 2 apresenta os fundamentos essenciais para a construção desta pesquisa. O Capítulo 3 lista os trabalhos considerados como de maior contribuição para a construção da pesquisa relatada nesta tese. O Capítulo 4 mostra o *worflow* criado nesta pesquisa para gerar modelos de uso dos dispositivos que compõem os smartphones baseados no padrão de comportamento dos usuários e descreve cada um dos passos adotados para cumprir esse objetivo. O Capítulo 5 apresenta o procedimento metodológico adotado para permitir implementar cada um dos passos descritos no *workflow* previamente gerado. O Capítulo 6 descreve as estratégias adotadas a fim de implementar os módulos de software criados nesta pesquisa para realizar a análise dos dados e geração dos modelos de aprendizagem de máquina de forma automatizada. No Capítulo 7, descrevemos a infraestrutura de hardware e software que foi fundamental para que esta pesquisa fosse realizada. No Capítulo 8, relatamos os resultados que obtivemos a partir dos modelos de aprendizagem de máquina construídos pelos módulos de software propostos nesta pesquisa. As conclusões, limitações e trabalhos futuros estão expostos no Capítulo 9.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta alguns dos principais conceitos empregados durante o desenvolvimento desta pesquisa. Iniciamos pelos conceitos de potência e energia e percorremos conceitos do Android, aprendizagem de máquina, otimização de hiperparâmetros, seleção de atributos, algoritmos genéticos e *drift* de conceito e dados.

2.1 POTÊNCIA E ENERGIA

Para efetuar o cálculo da potência instantânea durante a realização desta pesquisa, recorreu-se à relação existente entre potência, tensão e corrente mostrada na Equação 2.1.

$$P(t) = U(t) \times I(t) \tag{2.1}$$

Na Equação 2.1:

- **P(t)** representa a potência instantânea no tempo **t**,
- **U(t)** representa a tensão instantânea no tempo **t** e
- **I(t)** representa a corrente instantânea no tempo **t**.

Para que se pudesse calcular o consumo aproximado de uma amostra, fez-se necessário estabelecer um método que pudesse realizar uma aproximação para a integral mostrada na Equação 2.2, que representa a energia total consumida por um smartphone ao longo do tempo.

$$E(t) = \int_0^t P(t)\ dt \tag{2.2}$$

Para utilizar o conceito de integral nesta pesquisa recorremos à regra de Simpson.

2.2 CONCEITOS ANDROID

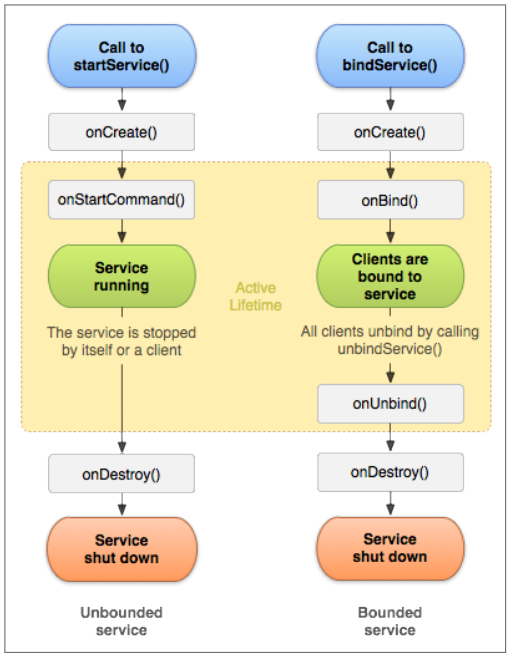
Para o desenvolvimento da aplicação responsável por coletar os dados de uso do usuário e de consumo do smartphone, se faz necessário entender alguns conceitos necessários para o desenvolvimento de aplicações e serviços Android.

2.2.1 Serviços

De acordo com Android Developers (DEVELOPERS, 2022c), um serviço é um componente de aplicação que pode executar operações por longo período de tempo e que não possui interface com o usuário. Há duas formas de iniciar um serviço Android: através de um componente de aplicação que pode iniciá-lo, ou através de um componente de aplicação que pode se ligar a ele para interagir com ele ou até mesmo fazer uma comunicação interprocessos.

O ciclo de vida de um componente de serviço Android é mostrado na Figura 2. Nela, como podemos verificar, após a chamada do método *startService*, o serviço executa até esse componente decidir terminar ou uma aplicação terminá-lo através da chamada do método *stopService*.

Figura 2 – Ciclo de vida de um serviço Android



Fonte: DEVELOPERS (2022c)

2.2.2 *Broadcasts*

Aplicativos Android podem enviar ou receber mensagens de *broadcast* do sistema Android ou de outros aplicativos Android (DEVELOPERS, 2022a). Essas mensagens de *broadcast* são enviadas quando um evento de interesse ocorre.

Dessa forma, aplicativos podem registrar-se para receber determinadas mensagens de *bro-*

adcast, e, quando um *broadcast* for enviado, o sistema operacional automaticamente roteia a mensagem para os aplicativos que se inscreveram a fim de receber certos tipos de mensagens de *broadcast* (DEVELOPERS, 2022a).

2.3 APRENDIZAGEM DE MÁQUINA

A ideia de aprendizagem de máquina se baseia no fato de existir algum algoritmo que seja capaz de ajudar a máquina a aprender através de um processo indutivo nos dados para, em seguida, ser capaz de induzir respostas a partir de novos dados. De acordo com MITCHELL(1997 apud GÉRON,2019), "alega-se que um programa de computador aprende com a experiência E em relação a algum tipo de tarefa T e alguma medida de desempenho P se o seu desempenho em T, conforme medido por P, melhora com a experiência E.".

Algoritmos de aprendizagem de máquina têm sido amplamente utilizados em diversas tarefas, que podem ser divididas em: **Preditivas** e **Descritivas**. Nas tarefas preditivas, os algoritmos são aplicados a um conjunto de dados rotulados para induzir um modelo preditivo capaz de predizer, para um novo objeto representado pelos valores de seus atributos preditivos, o valor do seu atributo resposta. Nessas tarefas, são utilizados algoritmos de aprendizagem de máquina que seguem o paradigma de aprendizagem supervisionada (FACELI et al., 2021).

Em tarefas descritivas, ao invés de predizer um valor, os algoritmos de aprendizagem de máquina extraem padrões dos valores preditivos de um conjunto de dados, sendo considerada uma aprendizagem não-supervisionada. São tarefas descritivas: agrupamento de dados e regras de associação (FACELI et al., 2021).

As tarefas preditivas se distiguem pelo valor do atributo resposta a ser predito: discreto, no caso dos algoritmos de classificação; e contínuo, no caso dos algoritmos de regressão. Durante o processo de aprendizado, um algoritmo de aprendizagem de máquina procura por um modelo, no espaço de possíveis modelos, capaz de modelar a relação entre os atributos preditivos e o atributo resposta (FACELI et al., 2021).

O formato usado por um algoritmo para representar os possíveis modelos define a preferência ou **viés de representação** do modelo, restringindo os modelos que ele pode encontrar. A forma como um algoritmo procura pelo melhor modelo, no espaço de possíveis modelos, define um outro viés, chamado **viés de busca** (FACELI et al., 2021).

Dessa forma, segundo FACELI et al.(2021), o viés é necessário para restringir os modelos a serem avaliados no espaço de busca, não tornando possível o aprendizado/generalização sem

a presença do viés.

Raj (2021) discute os princípios fundamentais adotados para implantação de modelos de aprendizagem de máquina. Ele começa discutindo a respeito do que devemos considerar quando estamos trabalhando em um ambiente de pesquisa e em um ambiente de produção. As diferenças são apontadas na Tabela 1.

Tabela 1 – ML na Pesquisa vs Produção

	Pesquisa	**Produção**
Dados	Estático	Dinâmico(constantemente mudando)
Equidade	Recomendado	Necessário
Interpretabilidade	Recomendado	Necessário
Desempenho	Estado da Arte	Melhor que os modelos simples
Prioridade	Treinamento rápido	Inferência rápida

Fonte: (RAJ, 2021)

Ao longo da nossa pesquisa, embora estejamos em um ambiente acadêmico, buscaremos adotar algumas características listadas por RAJ(2021) para o uso de modelos de aprendizagem de máquina em ambientes de produção no que tange às características referentes a: Interpretabilidade, Desempenho e Prioridade. Dessa forma, optamos por, durante essa pesquisa, priorizar os modelos baseados em árvores, modelos lineares e modelos baseados em distância. Esses modelos conseguem atender bem a esses três requisitos, falicitando a interpretabilidade, tendo desempenho superior aos modelos mais simples e possuindo tempo de inferência rápida. Após o advento da aprendizagem profunda, esses modelos passaram a ser chamados de modelos rasos(*shallow*).

A seguir discutiremos as classes de algoritmos mais importantes para a geração dos modelos de aprendizagem de máquina gerados nesta pesquisa.

2.3.1 Modelos Lineares para Regressão

Géron (2019) define um modelo de regressão linear como um modelo que realiza predições simplesmente computando uma soma ponderada dos atributos de entrada, mais uma constante chamada de **termo bias**. A Equação 2.3 mostra a forma vetorizada para o modelo de regressão linear.

$$\hat{y} = h_\theta(x) = \theta \cdot x \tag{2.3}$$

Na Equação 2.3:

- θ representa o vetor de parâmetros do modelo, contendo o termo viés θ_0 e os pesos dos n atributos que o modelo possa ter: θ_1 a θ_n.
- x é o vetor de atributos da instância, contendo x_0 a x_n, com x_0 sempre igual a 1.
- $\theta \cdot x$ é o produto escalar dos vetores θ e x.
- h_θ é a função hipótese, usando os parâmetros do modelo θ.

Para treinar um modelo de regressão linear, busca-se encontrar o valor de θ que minimize a métrica Mean Squared Error (MSE). A MSE de uma hipótese h_θ de Regressão Linear em um conjunto de treinamento **X** é calculada usando a Equação 2.4.

$$MSE(X, h_\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)}\right)^2 \tag{2.4}$$

Para encontrar o valor de θ que minimiza a função de custo MSE, usamos a Equação Normal(2.5), equação essa utilizada pelo processo de minimização dos erros quadráticos.

$$\hat{\theta} = \left(X^T X\right)^{-1} X^T y \tag{2.5}$$

Na Equação 2.5:

- X^T representa a matriz transposta da matriz dos valores de entrada X. Cada uma das colunas da matriz dos valores de entrada X contém cada um dos atributos a serem usados como preditores. Cada uma das linhas da matriz dos valores de entrada contém uma amostra usada como entrada do modelo.
- $\hat{\theta}$ é o valor de θ que minimiza a função de custo.
- y é o vetor de valores resposta contendo $y^{(1)}$ a y^m.

No modelo de regressão linear simples, normalmente, o único hiperparâmetro é **fit_intercept** para permitir que o **termo bias** seja calculado pelo modelo. No entanto, há outras versões de modelos lineares que usamos durante esta pesquisa, conforme listados a seguir.

1. Lasso : Lasso é um modelo linear que estima coeficientes esparços. Ele se torna um modelo bastante útil em muitos contextos por preferir soluções com poucos coeficientes não-zero, reduzindo efetivamente o número de atributos dos quais uma dada solução é dependente.

O termo de regularização *Lasso* é expresso pela Equação 2.7. A estimativa Lasso resolve a minimização da penalidade sobre a equação dos mínimos quadrados, expressa pela Equação 2.6, usando o termo $\alpha\|\theta\|_1$, em que α é uma constante e $\|\theta\|_1$ é a norma l1 do vetor de coeficientes (SCIKIT-LEARN, 2022b).

$$OLS = \frac{1}{2}\|X\theta - y\|^2 = \frac{1}{2}(X\theta - y)^T(X\theta - y) \tag{2.6}$$

$$min\frac{1}{2n_{samples}}\|X\theta - y\|_2^2 + \alpha\|\theta\|_1 \tag{2.7}$$

2. Ridge: A regressão *Ridge* aborda alguns dos problemas da equação dos mínimos quadrados, impondo uma penalidade na dimensão dos coeficientes. Dessa forma, os coeficentes *ridge* minimizam a soma dos quadrados residuais penalizados de acordo com a Equação 2.8 (SCIKIT-LEARN, 2022b). O parâmetro de complexidade $\alpha \geq 0$ controla a quantidade da redução, proporcionando que, ao aumentar α, o modelo se torne mais robusto à colinearidade.

$$min\|X\theta - y\|_2^2 + \alpha\|\theta\|_2^2 \tag{2.8}$$

3. Regressão Linear Generalizada: Segundo AMR(2022), a função sigmoid fornece valores entre 0 e 1, de forma semelhante a probabilidades, como expresso na Equação 2.9.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.9}$$

Dessa forma, se substituirmos a variável z da Equação 2.9 por uma equação de regressão linear, o resultado do nosso modelo de regressão estará no intervalo entre 0 e 1 e teremos a expressão final expressa pela Equação 2.10.

$$E(y|x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n) = \frac{1}{1 + e^{\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n}} \tag{2.10}$$

A função expressa pela Equação 2.10 é normalmente chamada de **função de ligação inversa**. Com isso, os modelos de regressão linear generalizados estendem os modelos lineares de duas formas:

1. Os valores preditos $\hat{y}$ são ligados a uma combinação linear dos valores de entrada **X** via uma função ligação inversa como $\hat{y}(\theta, X) = h(X\theta)$.

2. A função de perda quadrática é substituída pelo desvio unitário d de uma distribuição na família exponencial (um modelo de dispersão exponencial reprodutiva (EDM)). Dessa

forma, o problema de minimização passa a ser expresso pela Equação 2.11, em que α é a regularidade de penalização L2. Quando os pesos de uma amostra são fornecidos, a média se torna uma média ponderada.

$$min\frac{1}{2n_{samples}}\sum_{i} d(y_i, \hat{y}_i) + \frac{\alpha}{2}\|\theta\|_2^2 \tag{2.11}$$

4. Orthogonal Matching Pursuit: Essa estratégia implementa o algoritmo Orthogonal Matching Pursuit (OMP) para aproximar o ajuste feito por um modelo linear com limitações impostas no número de coeficientes não-zero de acordo com a Equação 2.12, com $\|\theta\|_0 \leq n_{naozero\_coefs}$ (SCIKIT-LEARN, 2022b).

$$argmin\|y - X\theta\|_2^2 \tag{2.12}$$

5. ElasticNet: É um modelo de regressão linear treinado com ambos os termos de regularização l1 e l2. Esta combinação permite a geração de um modelo esparço com poucos termos não-zero, como o usado na regularização *Lasso*, enquanto mantém as propriedades da regularização *Ridge*. A combinação convexa dos termos l1 e l2 é feita pelo termo ρ. Esse tipo de modelo é útil quando se tem múltiplos atributos correlacionados entre si. A função objetivo a ser minimizada nesses casos é dada pela Equação 2.13(SCIKIT-LEARN, 2022b).

$$min\frac{1}{2n_{samples}}\|X\theta - y\|_2^2 + \alpha\rho\|\theta\|_1 + \frac{\alpha(1-\rho)}{2}\|\theta\|_2^2 \tag{2.13}$$

6. Gradiente Descendente Estocástico(SGD): O gradiente descendente estocástico é uma abordagem eficiente para treinar modelos lineares quando se tem muitas amostras e muitos atributos no conjunto de treinamento(SCIKIT-LEARN, 2022b).

2.3.2 K Vizinhos Mais Próximos (KNN)

O algoritmo dos **K vizinhos mais próximos** é um exemplo de algoritmo baseado em instâncias, em que as amostras do conjunto de treinamento são armazenadas de modo tal que novas amostras são estimadas baseadas na similaridade com as amostras no conjunto de treinamento (FACELI et al., 2021).

O algoritmo dos vizinhos mais próximos tem variações definidas pelo número de vizinhos considerados. Quando se considera apenas 1 vizinho, cada objeto representa um ponto no

espaço definido pelos atributos. Com isso, calcula-se a distância entre dois pontos usando uma métrica de distância, sendo a mais comumente utilizada a distância euclidiana mostrada na Equação 2.14.

2.14.

$$d_{euclidean}(x, y) = \sqrt{\sum_i (x_i - y_i)^2} \tag{2.14}$$

Após definir uma função de distância para determinar quais as amostras mais similares em relação à nova amostra não vista, nós precisamos definir uma função que combine amostras similares para prover uma estimativa para a nova amostra. Uma das estratégias escolhidas pelo algoritmo regressor KNN é a **Função de Voto Ponderado**. Dessa forma, para classificar uma nova amostra, baseada em 2 vizinhos (A e B), nós devemos usar a relação expressa pela Equação 2.15 (LAROSE, 2014).

$$w = \frac{1}{d(new, A)^2} + \frac{1}{d(new, B)^2} \tag{2.15}$$

Então, o valor da variável resposta $\hat{y}$ é calculado como descrita por Larose (2014) e explicada pela Equação 2.16.

$$\hat{y} = \frac{\sum_i w_i y_i}{\sum_i w_i} \tag{2.16}$$

2.3.3 Árvores de Regressão

Uma árvore de decisão usa a estratégia de dividir para conquistar de modo a resolver um problema de decisão. Um problema complexo é dividido em problemas mais simples, aos quais recursivamente é aplicada a mesma estratégia. As soluções dos subproblemas são, então, combinadas na forma de uma árvore para resolver um problema complexo. A força dessa estratégia vem da capacidade de dividir o espaço de instâncias em subespaços, ajustando cada um deles e usando um modelo diferente (FACELI et al., 2021).

Uma árvore de regressão é definida formalmente como um grafo direcionado acíclico, em que cada nó ou é um nó de divisão, com dois ou mais sucessores, ou um nó folha. Usualmente, em um nó folha tem-se uma função na qual são considerados apenas os valores da variável resposta. Em problemas mais simples, a função usada é a função constante, que minimiza a função custo, a qual, em problemas de regressão, é a MSE. Já o nó de divisão contém um teste condicional baseado nos valores de um ou mais atributos (FACELI et al., 2021).

A construção de uma árvore de regressão leva em consideração a função custo dada pelo MSE dos valores resposta nas folhas. Para se fazer a divisão em um nó de divisão, leva-se em consideração a métrica Standard Deviation Reduction (SDR). Dessa forma, considerando um conjunto de dados **D** com **n** exemplos, a variância da variável resposta **y** é dada pela expressão da Equação 2.17.

$$sd(D, y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.17}$$

Para exemplificar o uso da métrica SDR,se considerarmos um teste h_A sobre um atributo **A**, expresso por $\mathbf{A} \leq a_1$, teremos os exemplos do conjunto **D** sendo divididos em dois subconjuntos $\mathbf{D}_L$ e $\mathbf{D}_R$, com tamanhos n_L e n_R, tais que $n = n_L + n_R$. O algoritmo procurará um valor a_1 para usar nesse teste h_A de tal modo que a variância de **y**, nos subconjuntos $\mathbf{D}_L$ e $\mathbf{D}_R$, seja sempre menor ou igual à variância de **y** após a divisão. Pode-se estimar a redução na variância obtida pela aplicação do teste h_A aplicando a relação expressa na Equação 2.18.

$$SDR(h_A) = sd(D, y) - \frac{n_L}{n} \times sd(D_L, y) - \frac{n_R}{n} \times sd(D_R, y) \tag{2.18}$$

Para cada atributo e teste no valor do atributo, é calculada uma redução da variância associada a esse teste. O teste que provoca maior redução é escolhido para ser o teste para o nó (FACELI et al., 2021).

2.3.4 *Ensembles*

De acordo com Wade e Glynn (2020), um método *ensemble* é um modelo de aprendizagem de máquina que agrega os resultados das predições de vários modelos, tornando-se, então, menos suscetível a erros. Para os classificadores, a opção padrão é usar a estratégia de voto majoritário, enquanto que, para regressão, usa-se normalmente a média das estimativas dos modelos individuais.

Os métodos *ensemble* se dividem comumente em dois tipos: os que combinam diferentes modelos e os que combinam muitas versões do mesmo modelo. O método que combina muitas versões do mesmo modelo é, novamente, dividido em dois tipos: os que combinam os modelos usando a estratégia de agregação por *bootstrapping*, chamados de *bagging*; e os que aprendem a partir dos erros dos modelos individuais, ajustando os novos modelos baseados nos erros dos modelos anteriores, chamados de *boosting* (WADE; GLYNN, 2020).

Nesta pesquisa trabalharemos com o modelos *ensemble* que utilizam muitas versões de um mesmo modelo. Tomamos essa decisão baseados no fato de que, ao criarmos muitas versões do mesmo modelo, construímos o que chamamos de aprendizes fortes. Com isso, trabalharemos com algoritmos que fazem bagging e boosting, conforme explicado a seguir.

1.Bagging: Os algoritmos que trabalham com a estratégia *bagging* são o Random Forest e o ExtraTrees. Conforme explicado anteriormente, a estratégia *bagging* funciona com *bootstrapping*, que significa amostragem com repetição(WADE; GLYNN, 2020).

No Random Forest, o *bootstrapping* ocorre quando cada árvore de regressão é construída. Se todas as árvores de regressão consistem das mesmas amostras, elas terão predições similares, fazendo o resultado agregado ter predição semelhante à árvore individual. Ao invés disso, com *random forest*, as árvores são construídas usando *bootstrapping*, normalmente com a mesma quantidade de amostras do conjunto de treinamento original(WADE; GLYNN, 2020).

2. Boosting: Exitem diversos algoritmos que trabalham com essa estratégia. Há duas principais estratégias de trabalho com *boosting* que merecem ser destacadas: *Adaboost* e *Gradient Boosting*(WADE; GLYNN, 2020).

O *Adaboost* é um dos mais antigos e populares modelos boosting. Nele, cada nova árvore ajusta seus pesos baseando-se nos erros das árvores anteriores, prestando mais atenção às predições feitas de forma errônea através do ajuste dos pesos que afetam essas amostras com um percentual maior. Dessa forma, o *Adaboost* transforma árvores com baixa capacidade preditiva em novas árvores com grande capacidade preditiva, através de um processo de correção de erros iterativo (WADE; GLYNN, 2020).

Já o *Gradient Boosting* utiliza uma abordagem diferente. Embora essa abordagem também realize ajustes nos pesos das amostras baseados no erros de predição, ela vai um passo adiante, treinando uma nova árvore inteiramente baseada nos erros de predição das árvores anteriores. Dessa forma, para cada nova árvore gerada, a abordagem de *gradient boosting* observa os erros e constrói essa nova árvore completamente baseada nesses erros (WADE; GLYNN, 2020).

2.4 OTIMIZAÇÃO DE HIPERPARÂMETROS

Segundo Zheng (2015), no domínio da aprendizagem de máquina, o ajuste de hiperparâmetros é uma tarefa de meta-aprendizagem. De acordo com a autora, modelos de aprendizagem de máquina são basicamente funções matemáticas que representam o relacionamento entre

diferentes aspectos dos dados.

Para citar um exemplo, em um modelo de regressão linear, que utiliza uma linha para representar o relacionamento entre os atributos e a variável resposta, existe uma equação como:$w^T x = y$, em que x é o vetor que representa o conjunto de atributos, e y é uma variável escalar que representa a resposta. A variável w para esse modelo representa os coeficientes dos termos em x – sendo w^T a matriz transposta da matriz de coeficientes w– e é conhecido como o parâmetro do modelo, que é ajustado ao longo do processo de treinamento (ZHENG, 2015). O processo de treinamento envolve usar um processo de otimização para determinar o melhor conjunto de parâmetros para um modelo que se ajuste aos dados.

Há, no entanto, um outro conjunto de parâmetros conhecidos como **hiperparâmetros** que devem ser especificados fora do processo de treinamento. Por exemplo, as árvores de decisão possuem hiperparâmetros tais como profundidade e número de folhas na árvore. Já os algoritmos de máquinas de vetores de suporte requerem a configuração do termo de penalização pelo erro e, para os algoritmos que possuem *kernel*, a configuração dos hiperparâmetros de *kernel*, como a largura do raio para a função de base radial.

As configurações de hiperparâmetros geram um grande impacto na acurácia de predição do modelo treinado. Para cada conjunto de dados a ser usado para treinamento, há um conjunto específico de hiperparâmetros e, como o processo de treinamento do modelo não configura valores para os hiperparâmetros, se faz necessário ter-se um metaprocesso que ajuste os hiperparâmetros.

O ajuste de hiperparâmetros é uma tarefa de meta-otimização. Cada tentativa de configuração de hiperparâmetros envolve treinar um modelo – um processo de otimização interno. Com isso, a saída do ajuste de hiperparâmetros é a melhor configuração de hiperparâmetros, e a saída do processo de treinamento do modelo é a melhor configuração dos parâmetros do modelo.

Zheng (2015) destaca que o ajuste de hiperparâmetros é uma tarefa de otimização, assim como o treinamento do modelo, mas se torna uma tarefa bem mais difícil porque, ao contrário do processo de treinamento que possui uma função de penalização para ajuste dos parâmetros, ela depende da saída de uma caixa preta (o processo de treinamento).

Diante dessa constatação precisamos adotar um algoritmo próprio para ajustar os hiperparâmetros dos algoritmos que utilizaremos durante esta pesquisa. Existem diversas técnicas para fazer a otimização de hiperparâmetros e a seguir explicaremos as técnicas mais usadas.

2.4.1 Estratégias tradicionais de otimização

As técnicas de otimização de hiperparâmetros tradicionais envolvem a utilização de uma estratégia baseada em tentativa e erro. Há três técnicas que utilizam essa estratégia(KUMAR, 2022):

Manual Search: Ao utilizar essa técnica, o desenvolvedor precisa tentar diferentes combinações de hiperparâmetros para o modelo de aprendizagem de máquina e selecionar aquele que tenha o melhor desempenho.

Grid Search: *Grid Search* pode ser definido como uma versão automatizada da otimização de hiperparâmetros utilizando *Manual Search*. No entanto, *Grid Search* não é amigável computacionalmente falando, uma vez que ela consome muito tempo para otimizar.

Random Search: O *Grid Search* tenta todas as combinações de hiperparâmetros, aumentando assim o tempo de computação. *Random Search* treina modelos com combinações de hiperparâmetros definidas de forma aleatória.

2.4.2 Otimização Bayesiana

YE(2022) afima que a otimização Bayesiana provê uma estrutura elegante para resolver problemas de maximização ou minimização de uma função objetivo que possui as seguintes características:

- Possui alto custo computacional para calcular. Dessa forma, o método de otimização deve funcionar com uma quantidade limitada de amostragens de entrada.
- A derivada da função é desconhecida.
- Temos que encontrar um mínimo ou máximo global utilizando um mecanismo que não fique preso a um mínimo ou máximo local.

Com isso, se tivermos uma **função objetivo** $c(x)$ que representa uma função de custo para um modelo de aprendizagem de máquina, a otimização bayesiana irá minimizar ou maximizar essa função usando uma abordagem chamada **otimização substituta**. No contexto em questão, uma **função substituta** é uma aproximação da **função objetivo**(YE, 2022).

Segundo YE(2022), a **função substituta** é formada por pontos amostrados previamente e, baseado na **função substituta**, podemos identificar quais são pontos de máximo ou mínimo

promissores. Baseados nessa informação, decidimos coletar mais amostras de pontos dessa região promissora e atualizar a **função substituta** de acordo. A cada iteração vamos utilizando a **função substituta**, aprendendo mais sobre a áreas de interesse pela amostragem e atualizando a **função substituta**.

A essência da modelagem e estatística Bayesiana é a atualização de uma crença a *priori*(prévia) baseada em uma nova informação para produzir uma crença a *posteriori*(posterior). Nesse caso, a **otimização substituta** realiza exatamente isso, podendo, então, ser melhor representada através dos sistemas bayesianos, fórmulas e ideias(YE, 2022).

Vale lembrar que o teorema de Bayes é uma abordagem utilizada para calcular a probabilidade condicional de um evento, como mostrado na Equação 2.19:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{2.19}$$

Na Equação 2.19:

- $P(A|B)$: Representa a probabilidade de o evento A ocorrer dado que o evento B ocorreu.
- $P(B|A)$: Representa a probabilidade de o evento B ocorrer dado que o evento A ocorreu.
- $P(A)$: Representa a probabilidade de o evento A ocorrer.
- $P(B)$: Representa a probabilidade de o evento B ocorrer.

De acordo com Browlee (2019), podemos simplificar esse cálculo removendo o valor de normalização $P(B)$ e descrevendo a probabilidade condicional como uma quantidade proporcional. Podemos fazer isso por não estarmos interessados em calcular uma probabilidade condicional específica, mas, ao invés disso, por querermos otimizar uma quantidade. Dessa forma, ficamos com a relação mostrada na Equação 2.20.

$$P(A|B) = P(B|A) \times P(A) \tag{2.20}$$

Quando estamos trabalhando com otimização Bayesiana, a probabilidade condicional que estamos calculando passa a ser referenciada como **posterior**, a probabilidade condicional reversa é chamada de **verossimilhança** e a probabilidade marginal é referida como **anterior**. Com isso, passamos a expressar a Equação 2.20 usando a Equação 2.21.

$$posterior = verossimilanca \times anterior \tag{2.21}$$

A **função substituta** – representada como uma distribuição de probabilidade, a **anterior**– é atualizada usando uma **função de aquisição**. Esta função é responsável por guiar a proporção de novos pontos a serem avaliados, em um *trade-off* entre *exploration* e *exploitation*(YE, 2022):

- *Exploitation*: Busca por uma amostra na qual o modelo prevê um bom valor para a **função objetivo**. No entanto, se continuarmos a fazer *exploitation* em uma região podemos ficar presos em um máximo ou mínimo local, tendo pouco ganho.
- *Exploration*: Busca por amostras em locais cujo o valor da **função objetivo** é possivelmente alto. Esse comportamento garante que a maior parte da região seja explorada, possibilitando encontrar o máximo ou mínimo global.

O algoritmo de Otimização Bayesiana pode, então, ser resumido pela sequência de passos(YE, 2022):

1. Inicializar a distribuição **anterior** expressa pela **função substituta**.
2. Escolher vários pontos x tais que a **função de aquisição** que opera na distribuição **anterior** atual seja maximizada.
3. Avaliar os pontos x utilizando a **função objetivo** e obter os resultados, y.
4. Atualizar a distribuição **anterior** com os novos dados a fim de produzir a **posterior**(que irá se tornar a **anterior** no próximo passo).
5. Repetir os passos 2-5 por várias iterações.

Após entendermos as técnicas mais usadas para fazermos otimização de hiperparâmetros, precisávamos avaliar qual a técnica mais efeitiva para realizar a otimização. Kraus (2022) realizou um experimento para comparar a acurácia dos modelos e o tempo gasto para ajustar os hiperparâmetros deles usando as três técnicas previamente estudadas: *GridSearch*, *Random Search* e Otimização Bayesiana. Para tal o autor utilizou conjuntos de dados para classificação gerados pela biblioteca Scikit-Learn(PEDREGOSA et al., 2011). O classificador usado durante os experimentos feitos por KRAUS(2022) foi o Light Gradient Boosting Machine (LGBM) e os hiperparâmetros ajustados foram:

1 taxa de aprendizagem 2 profundidade máxima 3 número de estimadores

4 número de folhas
5 tipo de *boosting*
6 fator de regularização
7 fração de atributos usado para treinar cada árvore

KRAUS(2022) encontrou as seguintes acurácias e tempos para ajustar os hiperparâmetros: 0.79 e 237 segundos, com *Grid Search*; 0.8 e 11 segundos, com *Random Search* e 0.84 e 15 segundos, com Otimização Bayesiana. Baseados nos resultados encontrados pelo autor, adotamos a estratégia de otimização Bayesiana para fazer os ajustes nos hiperparâmetros.

2.5 SELEÇÃO DE ATRIBUTOS

De acordo com Kuhn e Johnson(2019), as principais motivações para remover preditores devem ser mitigar um problema específico na interação entre preditores e um modelo, ou reduzir a complexidade do modelo. Algumas questões, como as listadas abaixo, são importantes de serem destacadas,caso não se faça seleção de atributos:

- Alguns modelos, como máquinas de vetor de suporte e redes neurais, são sensíveis a preditores irrelevantes.
- Outros modelos, como regressores lineares ou logísticos, são vulneráveis a preditores correlacionados. Desse modo, a remoção de preditores irá reduzir a multicolinearidade e, com isso, ajudar esses modelos a terem um melhor ajuste aos dados.
- Mesmo quando se tem modelos insensíveis a preditores extras, é salutar incluir o mínimo de preditores possível que forneça resultados aceitáveis. Em alguns casos, a redução na quantidade de preditores reduz o custo na aquisição de novos dados ou melhora o *throughput* do sistema usado para fazer predições.

Dessa forma, o principal objetivo de se fazer seleção de atributos pode ser formulado como: reduzir o número de preditores tão quanto possível sem comprometer a capacidade preditiva do modelo.

Segundo Zheng e Casari(2018), para obter um modelo com uma menor quantidade de preditores, algumas técnicas demandam o treinamento de mais de um modelo candidato. Em outras palavras, seleção de atributos não é sobre reduzir o tempo de treinamento dos modelos – de fato, algumas técnicas, como Eliminação Recursiva de Atributos e Algoritmos Genéticos, aumentam o tempo de treinamento –, mas sobre reduzir o tempo de estimativa do modelo.

Kuhn e Johnson (2019) classificam as abordagens usadas para seleção de atributos em:

1. Gulosa: A busca gulosa é uma abordagem que escolhe um caminho de busca baseado na direção que parece melhor no momento para atingir o melhor benefício imediato. Embora esta possa ser uma estratégia efetiva, pois pode mostrar benefícios imediatos no desempenho preditivo do modelo, ela pode ficar presa em um ótimo local, pois não avalia configurações com um desempenho preditivo momentaneamente pior.

2. Não-Gulosa: A abordagem não-gulosa reavalia combinações de atributos previamente testados e tem a habilidade de se mover em uma direção desfavorável inicialmente, caso ela aparente ter pontenciais benefícios após algumas iterações. Com isso, a estratégia não gulosa pode escapar de ficar presa em um ótimo local.

Zheng e Casari (2018) classificam as técnicas de seleção de atributos em:

1. Filtragem: As técnicas de filtragem pré-processam os atributos para remover aqueles menos prováveis de serem úteis para o modelo. Por exemplo, pode-se usar uma técnica de informação mútua entre cada atributo e a variável resposta, e filtrar aqueles que estiverem abaixo de um limiar. Essas técnicas possuem um custo computacional menor que os métodos *wrapper*, com a desvantagem de não levarem em consideração o algoritmo a ser empregado e, consequentemente, podem não selecionar os atributos mais adequados a um algoritmo.

2. Métodos Wrapper: São técnicas com custo computacional elevado , mas permitem ao projetista tentar subconjuntos de atributos, significando que o projetista não irá acidentalmente remover atributos que não são informativos por si só mas são úteis quando considerados em conjunto.

3. Métodos Embutidos: Estes métodos fazem seleção de atributos como parte do processo de treinamento do modelo. Por exemplo, um algoritmo de árvore de decisão realiza a seleção de atributos de forma inerente porque ele seleciona um atributo no qual ele possa particionar a árvore a cada passo do treinamento. Esses métodos não são tão poderosos quanto os métodos *wrapper*, mas também não são computacionalmente tão custosos. Dessa forma, os métodos de seleção de atributos embutidos encontram um equilíbrio entre o custo computacional e a qualidade dos resultados.

Um exemplo de um método *wrapper* que possui uma abordagem gulosa é o método de retrosseleção de atributos (também conhecido como Eliminiação Recursiva de Atributos)(KUHN; JOHNSON, 2019). Nesse método, os preditores são inicialmente classificados por alguma me-

dida de importância. Um modelo com todos os atributos é inicialmente criado, em seguida ele passa a ser baseado em um conjunto menor de atributos no qual os atributos menos importantes são removidos. Este processo continua removendo os atributos baseados em suas classificações, até que o modelo permaneça com um pequeno número de atributos.

Exemplos de métodos não-gulosos são algoritmos genéticos e têmpera simulada. O método baseado em têmpera simulada é não-guloso uma vez que ele incorpora aleatoriedade no processo de seleção de atributos.

2.6 ALGORITMOS GENÉTICOS

Os algoritmos genéticos fazem parte de uma classe de algoritmos chamados de algoritmos evolucionários. Segundo Kopec e Safari (2019), os algoritmos genéticos são utilizados quando abordagens algorítmicas tradicionais são insuficientes para se chegar à solução de um problema em tempo razoável.

De acordo com Kopec e Safari (2019), os algoritmos genéticos se baseiam na teoria da evolução, utilizando os conceitos de **indivíduos** e **população** e as operações de *crossover*, **mutação** e **seleção de indivíduos**. A seleção de indivíduos ocorre segundo uma **função de aptidão** (fitness function).

Dessa forma, um algoritmo genético possui uma população de indivíduos chamados **cromossomos**. Os cromossomos são compostos de **genes** que especificam as características e competem para resolver alguns problemas. A competência que um cromossomo tem para resolver um problema é definida pela função de aptidão (função de fitness).

Um algoritmo genético executa através de gerações, ou seja, a cada iteração do algoritmo todos os cromossomos são substituídos por novos cromossomos. Em cada geração, há uma maior chance de os melhores cromossomos serem selecionados para reproduzir-se. Esses cromossomos selecionados possuem uma chance de realizar a troca (crossover) de genes entre si, e cada cromossomo tem uma probabilidade de **mutar** algumas de suas características.

Eiben e Smith (2015) definem que os componentes mais importantes de um algoritmo evolucionário – a classe de algoritmos da qual os algoritmos genéticos fazem parte – são:

1. Representação (definição dos indivíduos): O primeiro passo na definição de um algoritmo evolucionário é estabelecer uma ligação entre o mundo real (universo do problema), isto é, estabelecer uma ponte entre o contexto do problema original e o espaço de resolução

do problema no qual o algoritmo executa. Eiben e Smith (2015) dizem que os objetos que formam as possíveis soluções do problema original são chamados **fenótipos**, enquanto as suas respectivas codificações para o espaço de solução do algoritmo evolucionário são chamadas de **genótipos**.

2. Função de aptidão (função de fitness): O papel da **função de aptidão** é especificar os requisitos que a população deve se adaptar para atender. Dessa forma, ela estabelece a base para que a seleção dos indivíduos ocorra, facilitando o processo de melhoria. Tecnicamente falando, essa função é responsável por atribuir uma medida de qualidade ao genótipo.

3. População: O papel da população é manter o conjunto das possíveis soluções. Uma população é, dessa forma, um conjunto de genótipos. A população forma a unidade de evolução, uma vez que, como os indivíduos são objetos que não mudam ou se adaptam, quem evolui é a população. Dada uma determinada representação do problema, uma população pode ser tão simples quanto definir a quantidade de indivíduos, **tamanho da população**, que estão dentro dela.

4. Mecanismo de seleção parental: O papel da **seleção parental** é distiguir entre indivíduos, baseando-se nas suas qualidades, e, em particular, permitir que os melhores indivíduos se tornem pais da próxima geração.

Um indivíduo é considerado um **pai** caso ele seja selecionado para sofrer variação com o intuito de criar filhos. Juntamente com o mecanismo de seleção de sobreviventes, a seleção parental é responsável por provocar as melhorias de qualidade. Em computação evolucionária, o mecanismo de seleção parental é frequentemente probabilístico, com indívíduos que possuem uma maior aptidão tendo mais chances de serem selecionados.

5. Mecanismos de Variação (Mutação e Recombinação): O papel dos operadores de variação é criar novos indivíduos a partir dos indivíduos mais velhos. No espaço de soluções fenotípicas, isso equivale a gerar novas soluções candidatas.

Os operadores de variação em computação evolucionária se dividem em dois tipos baseados em suas aridades: variações unárias (mutação) e variações n-árias (recombinação ou crossover).

6. Mecanismo de seleção de sobreviventes (Substituição): Similar ao mecanismo de seleção parental, o papel da **seleção de sobreviventes** é distiguir entre os indivíduos baseados nas suas respectivas qualidades.

Entretanto, o mecanismo de seleção de sobreviventes é utilizado em um estágio diferente do ciclo evolucionário – o mecanismo de seleção de sobreviventes é chamado logo após a cria-

ção dos filhos a partir dos pais selecionados. Em contraste ao mecansimo de seleção parental que é um processo estocástico, o mecanismo de seleção de sobreviventes é um mecanismo determinístico que se baseia, frequentemente, na idade.

7. Inicialização: Frequentemente, a estratégia usada para inicializar um algoritmo evolucionário, gerando a primeira população, é feita de forma simples utilizando um mecanismo de geração de indivíduos aleatório.

8. Condição de Término: As condições de término podem ser distiguidas em dois casos: 1) quando temos um problema relativamente simples e sabemos qual o nível ótimo da nossa solução, podemos estabelecer como condição de parada o nível ótimo sabido previamente e 2) quando não temos ciência de qual o nível ótimo queremos atingir, normalmente estabelecemos como condição de parada um determinado número de iterações. Durante a realização desta pesquisa, estabelecemos como condição de parada 300 execuções do algoritmo.

2.7 *DRIFT* DE CONCEITO E DADOS

Após o treinamento de um modelo de aprendizagem de máquina, quando ele é posto em um ambiente de produção, ao longo do tempo, ele degrada a capacidade prediditiva dele devido à existência de *drift* de conceito ou *drift* de dados (TANNOR, 2022).

Ao entrar em produção, um modelo de aprendizagem de máquina está constantemente recebendo novos dados para realizar predições sobre eles. No entanto, esses novos dados podem ter uma distribuição de probabilidade diferente daquela existente nos dados de treinamento do modelo.

Na construção de módulos de *software* baseados em modelos de aprendizagem de máquina, "conceito"se refere à distribuição de probabilidade conjunta dos atributos de entrada(X) e do atributo resposta(Y), sendo expressa pela Equação 2.22. O *shift* de conceito ocorre quando a distribuição de probabilidade conjunta muda. Dessa forma, $P_{t1}(X, Y) \neq P_{t2}(X, Y)$, ou seja, a distribuição de probabilidade conjunta em um instante de tempo t_1 muda ao longo do tempo e se torna diferente em um outro instante de tempo t_2(TANNOR, 2022).

$$P(X, Y) = P(Y)P(X|Y) = P(X)P(Y|X) \tag{2.22}$$

A partir dessas definições, podemos definir *drift* de dados e conceito como:

1. Drift de dados: Ocorre quando a distribuição de probabilidade dos dados de entrada do modelo muda ao longo do tempo, ou seja, $P_{t1}(X) \neq P_{t2}(X)$.

2. Drift de conceito: Representa a situação em que o relacionamento funcional entre os dados de entrada do modelo de aprendizagem de máquina e os dados de saída do modelo mudam.

2.8 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a fundamentação teórica que possibilita o entendimento dos conceitos mais fundamentais para a compreensão das abordagens de aprendizagem de máquina e o desenvolvimento de aplicações móveis utilizadas para desenvolver os módulos de *software* necessários para construir os modelos de aprendizado de máquina que relacionem o padrão de comportamento do usuário e a utilização dos dispositivos que compõem o smartphone – atingindo, assim, o objetivo desta pesquisa.

Apresentamos conceitos importantes a respeito de consumo e energia, conceitos para o desenvolvimento de aplicações Android, aprendizagem de máquina, otimização de hiperparâmetros dos modelos, seleção de atributos, algoritmos genéticos e drift de conceito e dados.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma síntese dos trabalhos que contribuíram com a pesquisa relatada nesta tese. Foram realizadas buscas por trabalhos que retratassem o estado da arte em três momentos distintos durante a realização desta pesquisa.

3.1 MODELAGEM DO CONSUMO COM TÉCNICAS DE ENGENHARIA DE SOFTWARE

Inicialmente, investigamos trabalhos que buscassem modelar o consumo energético de smartphones, uma vez que um dos objetivos estabelecido nesta pesquisa foi criar modelos que relacionem o padrão de comportamento do usuário com a utilização dos dispositivos que compõem o smartphone. Muitos pesquisadores realizam estudos com o objetivo de modelar o consumo de energia do smartphone usando técnicas de engenharia de software.

A estratégia proposta por ROMANSKY et al.(2017) modela o consumo de energia do smartphone usando as chamadas do sistema dentro do código do aplicativo como entradas para uma Rede Neural Long Short Term Memory (LSTM). A abordagem desenvolvida pelos autores ajuda a identificar e resolver *bugs* de energia do software – erros de codificação do software que culminam no aumento do consumo energético do smartphone – alinhando *traces* do comportamento do software com *traces* de consumo de energia.

Os autores usaram 6 aplicativos Android em suas diferentes versões para coletar as chamadas do sistema e fornecê-las à Rede Neural, com o intuito de prever o consumo de energia do smartphone.ROMANSKY et al.(2017) desenvolveram uma metodologia para avaliar os modelos de consumo e comprovaram a eficácia de seu método, obtendo um erro de apenas 4% entre o consumo de energia medido e a previsão do modelo.

A eficiência dos modelos propostos pelos autores é bastante elevada. No entanto, é possível observar que há dois fatores que dificultam o uso da abordagem proposta por eles: 1) necessariamente, o código da aplicação deve ser instrumentalizado para construir os *traces* de execução da aplicação e 2) o desenvolvedor da aplicação precisa construir uma infraestrutura adicional, chamada GreenMiner, para coletar os dados de consumo energético da aplicação.

Nesta pesquisa, buscamos superar esses dois desafios estabelecendo uma abordagem que não necessite instrumentalizar o código fonte da aplicação, bem como não precise de um *hardware* adicional, mesmo não atingindo o mesmo nível de acurácia nos modelos.

Seguindo uma abordagem semelhante, alguns autores da área de pesquisa do modelo de consumo de energia de smartphones buscam desenvolver aplicativos *profilers* que instrumentalizem o código-fonte dos aplicativos Android estudados para exercitá-los e coletar dados estatísticos de uso da bateria (NUCCI et al., 2017)(HAO et al., 2013)(PATHAK et al., 2011).

NUCCI et al. (2017) apontam a necessidade de criação de uma estratégia de modelagem energética baseada em software, destacando que as ferramentas baseadas em hardware são muito precisas mas possuem como grande limitação o custo de aquisição do hardware específico. Os autores ainda destacam que muitas das abordagens baseadas em software para modelar o consumo energético dos smartphones se baseiam em medições de bateria assistidas por hardware. Dessa forma, os autores propõem uma ferramenta, baseada em software, de nome PETRA e comparam com a acurácia de modelagem dada por uma ferramenta de hardware, de nome Monsoon.

A ferramenta proposta por NUCCI et al. (2017) é composta de três principais blocos:

1. Processamento de aplicativo: Nessa etapa, a ferramenta PETRA precisa configurar o ambiente de execução antes de medir o consumo da aplicação. Para isso, é usada uma versão executável do aplicativo em formato Application Package (APk). Então, a ferramenta PETRA instala a aplicação e habilita a opção *debuggable*. Essa operação se faz necessária porque, de outro modo, ela não consegue instrumentalizar o aplicativo para medir o consumo energético.

2. Computação do perfil energético: Nessa etapa, o sistema PETRA exercita a aplicação, dando como entrada um caso de teste. Esse caso de teste pode ser criado usando uma ferramenta de automação como o *MonkeyRunner* ou com operações manuais feitas pelo engenheiro de software. Para fazer o perfil energético da aplicação, a ferramenta proposta utiliza os dados de uso dos dispositivos obtidos a partir da instrumentalização do aplicativo e dos dados de energia do arquivo *powerProfile*, fornecido pelo Android.

3. Geração da Saída: O arquivo final provido pelo sistema PETRA é um arquivo csv, contendo a estimativa de consumo para cada chamada de método da aplicação.

De acordo com os autores, o sistema proposto comete um erro menor que 0.001, o que corresponde a um consumo de $3.1 \times 10^{-6}\%$.

Já HAO et al.(2013) propuseram uma nova abordagem leve, em termos de requisitos de desenvolvimento, que provê estimativas de consumo energético bem precisas em nível de

código, usando uma abordagem que combina a análise do programa com modelagem energética por instrução.

A abordagem proposta pelos autores é composta de três entradas:

1. Gerador de carga responsável por traduzir a carga de trabalho em um conjunto de caminhos de execução de chamadas de função do software.

2. O analisador, que usa os caminhos de execução e os perfis do sistema para computar a estimativa de consumo.

3. Anotador de código que combina os caminhos de execução e as estimativas de consumo, gerando uma versão do código fonte que é provida ao desenvolvedor.

A abordagem proposta pelos autores consegue atingir um nível de precisão de 90% para um conjunto de aplicações da Google Play Store.

PATHAK et al. (2011) propuseram uma abordagem de modelagem de consumo baseada em chamadas de sistema que envolvem tanto o comportamento energético baseado na utilização de dispositivos do smartphone, quanto o comportamento interno do sistema operacional que consome energia sem que necessariamente um dispositivo que compõe o smartphone seja utilizado. A abordagem proposta pelos autores foi construída a partir das seguintes observações feitas por eles:

1. Chamadas de sistema representam a única forma pela qual as aplicações ganham acesso ao sistema, uma vez que seus nomes e parâmetros indicam qual dispositivo será usado e com qual intensidade.

2. Usando as chamadas de sistema como gatilhos para transições de estado de potência, naturalmente resolve uma das limitações da modelagem de potência baseada na utilização. Essa limitação ocorre devido ao fato de muitas chamadas de sistema que não demandam naturalmente uma alta utilização de componentes configurarem os respectivos componentes em estado de alto ou baixo consumo, gerando dúvidas a respeito de em que parte do sistema está a otimização – software ou hardware.

3. De forma similar, a invocação de chamadas de sistema que ligam ou desligam componentes de alto consumo imediatamente dispara mudanças no consumo desses componentes, evitando o atraso devido à amostragem periódica nos contadores no qual a modelagem baseada em utilização se funda.

4. Usar chamadas de sistema como gatilhos naturalmente sugere usar uma máquina de estados finitos anotada com eventos de tempo e carga de trabalho recentes para modelar as transições de estado.

5. Chamadas de sistema podem ser facilmente rastreadas para se obter as sub-rotinas, as *threads* e os processos responsáveis por ela.

Para avaliar a metodologia proposta, os autores projetaram um conjunto de testes de *benchmark*, chamado CTester, que inclui uma aplicação para cada componente cuidadosamente projetada para exercitar todas as chamadas de sistema relevantes. Além dessas aplicações, os autores projetaram uma aplicação responsável por executar as aplicações do *benchmark* desenvolvida de forma individual em tempos pré-determinados de modo a criar cenários de chamadas de sistemas concorrentes em um mesmo ou em múltiplos componentes.

Após a análise desses trabalhos, percebemos alguns pontos a serem melhorados na análise de consumo energético dos aplicativos: precisamos encontrar uma estratégia que não dependa da instrumentalização dos aplicativos; não podemos considerar como estratégia de medição de consumo o arquivo *powerProfile* fornecido pelo Android porque, sem a instrumentalização do aplicativo, não sabemos com precisão quanto tempo cada dispositivo foi usado e em qual estado de uso cada um desses dispositivos estava.

3.2 CATEGORIZAÇÃO DE APLICAÇÕES POR USO

MEHROTRA et al. (2021) propuseram uma abordagem de aprendizado supervisionado para prever o consumo energético das aplicações, dividindo-as, com base nesse critério, em três categorias: baixo, médio e alto. Para desenvolver esse modelo de predição, os autores monitoraram o consumo energético de 90 aplicações em 414 casos de teste.

O consumo energético foi registrado em diferentes graus de granularidade, variando desde o nível da aplicação como um todo até o nível do consumo gerado pelo display LCD, origem da comunicação e processador. O principal objetivo dos autores foi encontrar as aplicações com alto consumo energético e, consequentemente, as principais responsáveis pela drenagem da bateria. A medição de consumo foi feita usando a ferramenta *PowerTutor*. Um conjunto de dez atributos preditores que afetam o consumo energético foi considerado. O classificador baseado no algoritmo *Random Forest* apresentou o melhor desempenho, chegando a obter

97% de acurácia, significando um acerto de predição na quase totalidade das amostras usadas para teste.

Observando o trabalho feito por MEHROTRA et al. (2021), percebemos a importante contribuição dada à área, devido ao fato de a proposta dos autores já nos fornecer de forma automatizada e não-intrusiva uma excelente estimativa do consumo energético. No entanto, em nossa proposta, procuramos estabelecer também uma relação entre a aplicação e a forma como ela é usada.

Outra estratégia utilizada para construir o modelo de consumo de energia de um smartphone foi proposta por ALAWNAH; SAGAHYROON (2017), que ofereceram um modelo de consumo de energia usando o uso do usuário como entrada para uma Rede Perceptron Multicamada. Os autores selecionaram uma amostra de dez estudantes universitários e monitoraram a atividade deles nos seus respectivos smartphones Sony Acro S, com a ajuda de uma aplicação de registro de uso baseado em Android.

Os autores escolheram esse modelo devido ao fato de ele vir equipado com um sensor de alta precisão de interface com a bateria. Para avaliar a acurácia dos modelos, os autores dividiram a base de dados em 70% para treinamento, 15% para validação e 15% para teste, obtendo um Root Mean Squared Error (RMSE) da ordem de $0,20$. Esse resultado indica que o modelo treinado teve uma boa capacidade de generalização.

O trabalho proposto pelos autores está relacionado ao uso de hardware de alta precisão para obter energia instantânea, pois seu modelo está sujeito a flutuações em sua precisão devido à falha de hardware. Além disso, os autores não detalham como chegaram aos resultados mostrados no artigo, se atendo mais à análise das arquiteturas de redes neurais utilizadas.

Muitos pesquisadores realizam estudos usando aprendizado de máquina e redes neurais para identificar o contexto e prever a atividade do usuário usando classificadores.

XIE; ZHENG; ZHANG (2017) usam estratégias de aprendizado de máquina com o intuito de identificar o contexto de uso de uma plataforma de *e-learning* para adaptar o conteúdo servido aos alunos. A ferramenta desenvolvida utiliza sensores como acelerômetro, luz e som para determinar o contexto de uso.

Os autores usaram uma ferramenta, chamada *Skyclass*, desenvolvida por eles, que permite aprendizes assistirem a vídeos através de seus clientes móveis. O processo de reconhecimento do contexto de uso se dá através de um processo padrão de aprendizagem supervisionada. Quando os estudandes assistem aos vídeos, a leitura dos sensores é feita e processada usando estratégias de limpeza de dados, extração e seleção de atributos.

Os resultados obtidos pelos autores mostram que os melhores classificadores são aqueles baseados em árvores e *ensembles* de árvores por possuírem uma alta acurácia e um baixo consumo energético.

O trabalho realizado por XIE; ZHENG; ZHANG (2017) nos auxiliou durante a realização desta pesquisa porque fomos capazes de validar nossa estratégia de escolha dos algoritmos de aprendizagem mais apropriados. Os resultados obtidos por nós, em grande parte dos cenários de avaliação, nos indicou os algoritmos baseados em árvores como melhores preditores.

DAI; HO; RUDZICZ (2015) usam algoritmos de aprendizado de máquina para prever a atividade realizada por um usuário. Para atingir esse objetivo, os autores utilizam a técnica de transferência de atividades entre usuários que utiliza ações semelhantes realizadas por outros usuários para aumentar a capacidade do algoritmo de aprendizado de máquina de prever a atividade realizada por um usuário.

Desse modo, os autores propõem uma métrica híbrida de similaridade fazendo uso da *eigen-behavior distance* – uma métrica de distância entre as matrizes que representam os comportamentos de dois usuários distintos – e de uma análise de comportamento comum. O *framework* proposto pelos autores consiste em oito passos:

- Construir representações de atividades para todos os usuários de origem e destino.
- Construir o *eigen-behavior* – matriz de autovetores– de atividades do usuário alvo.
- Projetar todas as matrizes de comportamento no espaço vetorial de matrizes de autovetores de comportamento do usuário alvo.
- Computar a similaridade entre as atividades do usuário alvo e cada um dos usuários analisados (ambas as atividades semelhantes e diferentes) no espaço vetorial de matrizes de autovetores usando a *eigen-similarity*.
- Mensurar a similiaridade de comportamento baseado na análise de comportamento comum.
- Calcular o *score* de similaridade conjunta baseado na *eigen-similarity*.
- Transferir as instâncias dependentes do tempo via análise de transferência de componentes.
- Predizer as atividades do dia seguinte.

Os autores alcançaram resultados da ordem de 90% de acurácia de predição para algumas atividades, utilizando a abordagem proposta por eles, ao passo que as outras abordagens usadas para comparação atingem acurácia da ordem de 87%.

Com o trabalho realizado por DAI; HO; RUDZICZ (2015), pudemos averiguar estratégias para predizer a atividade do usuário, o que nos foi útil para avaliar os preditores usados pelos autores durante o projeto da estratégia proposta por eles.

BETTINI; CIVITARESE; PRESOTTO (2020) construíram um sistema, CAVIAR (Context-aware ActiVe and Incremental Activity Recognition), que combina aprendizagem semissupervisionada e raciocínio semântico sensível ao contexto. Os autores propõem o sistema para prever a atividade do usuário, aplicando raciocínio semântico em um classificador incremental.

Desse modo, o modelo de reconhecimento de atividades é continuamente atualizado usando aprendizagem ativa. Os resultados apontados pelos autores em um conjunto de dados real de 26 usuários mostram um aumento da taxa de reconhecimento, estendendo o número de atividades reconhecidas e, mais importante, reduzindo o número de consultas disparadas pelo processo de aprendizagem ativa. Com isso, o sistema proposto pelos autores tem um percentual de consultas de apenas 6% e uma acurácia de 88%.

A pesquisa realizada por BETTINI; CIVITARESE; PRESOTTO (2020) nos foi útil para buscarmos adotar uma abordagem de aprendizagem incremental em que iniciamos com um modelo construído a partir de um dia de uso do usuário e fomos, ao longo do tempo, avaliando a capacidade preditiva dos modelos gerados, retreinando-os à medida que temos mais dados à disposição.

SARKER; ABUSHARK; KHAN (2020) formulam o problema de predição de uso de aplicativos no smartphone baseados no contexto utilizando técnicas de *machine learning*. Para isso, os autores apresentam uma técnica efetiva de Principal Component Analysis (PCA) fundamentada em um modelo de predição de usos de aplicativos no smartphone baseado no contexto.

No modelo proposto pelos autores, denominado ContextPCA, eles, primeiro, processam os dados brutos de uso de aplicativos dos usários individuais, capturando dados ausentes, codificando os dados e escalonando os dados para uma análise futura.

Em seguida, os autores extraem os atributos contextuais do conjunto de treinamento usando PCA e geram uma quantidade de componentes principais com uma dimensão menor que a original. Uma vez processados os contextos de uso nos componentes principais, os autores constroem uma árvore de decisão para, assim, atingirem o objetivo desejado.

O trabalho proposto por SARKER; ABUSHARK; KHAN (2020) nos foi relevante durante essa

pesquisa para testarmos estratégias que fossem capazes de avaliar diversos subconjuntos de atributos e assim escolher o mais apropriado.

Em outro trabalho, XIA et al. (2020) construíram um modelo de predição de uso de aplicativos utilizando técnicas de aprendizagem profunda para uma grande quantidade de usuários. Eles propuseram um *framework* de aprendizado multitarefa denominado *DeepApp*, com a finalidade de prever o próximo aplicativo a ser usado a partir do histórico de uso dos aplicativos por parte do usuário.

Para lidar com a escassez de dados, os autores treinaram uma rede neural geral para múltiplos usuários com dados históricos de usos insuficientes, a fim de compartilhar os padrões de uso comuns. De acordo com os autores, os modelos propostos têm uma acurácia 6.44% superior às estratégias propostas pelo estado da arte.

Embora o objetivo desta pesquisa não seja prever o uso do próximo aplicativo a ser usado, a abordagem proposta por XIA et al. (2020) é bastante interessante porque propõe o uso de uma rede neural geral que retrata o padrão de muitos usuários e, com isso, compartilha padrões comuns.

3.3 MODELAGEM E MELHORIA DO CONSUMO DE ENERGIA DO SMARTPHONE

Alguns pesquisadores têm realizado estudos buscando entender a relação entre o padrão de uso dos usuários e o consumo energético dos smartphones por eles utilizados.

DING; WANG; WANG (2021) realizaram uma pesquisa para avaliar a relação entre a interação do usuário e o consumo energético do smartphone. De acordo com os autores, é possível explorar o comportamento do usuário a partir da perspectiva dos dados de consumo energético. Segundo eles, a construção dessa relação entre a interação do usuário e o consumo energético do smartphone pode reduzir o custo de aquisição de dados para os fabricantes desses dispositivos e, ainda mais importante, pode permitir múltiplas aplicações úteis, tais como: localizar componentes que precisam ser atualizados mais urgentemente por drenar mais a bateria; assumindo que o comportamento dos usuários não muda muito entre as gerações de smartphones, pode-se estimar o desempenho energético da próxima geração de smartphones baseado no padrão de uso da geração atual.

Os autores propõem, então, uma abordagem de modelagem do padrão de uso do usuário baseado no consumo energético do smartphone chamada *E-Sub*. Especificamente, os modelos de comportamento dos usuários são extraídos a partir dos dados de consumo energético de

cada um dos dispositivos presentes no smartphone. Usando essa estratégia, os autores relatam que atingiram $93,97\%$ de cobertura do padão de uso com novos usuários no conjunto de teste.

A abordagem proposta por DING; WANG; WANG (2021) é bastante interessante, porém há alguns pontos que merecem ser analisados quanto aos módulos propostos por eles.

1. Em um dos módulos, eles fazem uma relação entre os aplicativos usados e o aumento de energia consumida pelo smartphone devido ao uso desses aplicativos. No entanto, fizemos experimentos durante esta pesquisa que mostram que, mesmo usando o mesmo aplicativo, temos padrões de consumo distintos a depender do padrão de uso dos usuário. Elementos como brilho da tela, tipo de conexão e preferências do usuário aumentam ou diminuem o consumo energético do smartphone.

2. Os autores consideram o consumo energético do smartphone através da soma dos percentuais de energia consumida por cada dispositivo, desacoplando o modelo energético do contexto situacional no qual o smartphone é utilizado.

Dessa forma, as ideias propostas por DING; WANG; WANG (2021) são bastante úteis mas precisam ser adequadas ao contexto no qual o smartphone é utilizado. Levando-se em consideração que, ao longo do tempo, devido a fatores externos, há uma variabilidade do consumo energético dos dispositivos, é muito mais útil adotar uma estratégia de aprendizado incremental, como a proposta por BETTINI; CIVITARESE; PRESOTTO(2020).

Outros pesquisadores realizam estudos para modelar e melhorar a eficiência energética dos smartphones com base no comportamento do usuário.

De acordo com TARKOMA et al. (2014), muitos estudos sobre o comportamento dos usuários em relação ao carregamento do seus smartphones foram realizados até o momento. Todos concordam em alguns pontos importantes, como, por exemplo, o de que a recarga é acionada pelo nível atual da bateria ou pelo contexto, que é derivado da hora e/ou localização.

TARKOMA et al. (2014) também discutiram até que ponto os aplicativos que informam como a bateria está sendo consumida pelo smartphone mudam o comportamento do usuário citando o exemplo de ATHUKORALA et al.(2014), que entrevistou 1.140 usuários do Carat perguntando como o aplicativo mudou seu comportamento. Os resultados mostraram que os usuários que usaram o aplicativo por mais de três meses foram muito mais propensos a deixar de usar aplicativos identificados como consumidores altos pelo aplicativo Carat.

LI; LIU; MEI(2018) propuseram uma estratégia baseada em rastreamentos de uso de smartphones para prever a duração da bateria usando modelos de aprendizado de máquina para previsão

da duração da bateria. Durante a pesquisa, os autores usaram o *Sherlock Data Set*, que é uma coleção de dados de vários anos mantida pelo BGU Cyber Security Research Center. Os autores alegaram que o tempo restante da bateria pode ser previsto com precisão usando a pesquisa deles.

Ainda assim, os autores reconhecem algumas limitações que podem afetar a generalização de seus resultados. O conjunto de dados é coletado de um único modelo de dispositivo e os usuários, em seu conjunto de dados, são principalmente de um grupo de usuários controlado de Israel, o que pode afetar o padrão de uso. Embora essa constatação feita pelos autores seja válida para qualquer grupo de usuários controlados, os autores quiseram destacar essa limitação presente no trabalho deles.

A pesquisa realizada por LI; LIU; MEI (2018) nos trouxe diversas ideias a respeito de como poderíamos construir modelos de consumo baseados em registros de uso do usuário. No entanto, desenvolvemos um método mais geral que não depende da localização do usuário durante esta pesquisa.

YAN; TAN; FU(2019) propuseram uma estrutura de gerenciamento de energia consciente do comportamento do usuário que pode otimizar as frequências da CPU para todos os diferentes tipos de comportamento.

Os autores classificam os tipos de comportamento do usuário em:

- baixo percentual de conteúdo atualizado e baixa frequência de entrada;
- baixo percentual de conteúdo atualizado e alta frequência de entrada;
- alto percentual de conteúdo atualizado e baixa frequência de entrada e
- alto percentual de conteúdo atualizado e alta frequência de entrada.

Baseados nessas categorias, os autores propõem um *framework* que ajusta a frequência da CPU de acordo com a categoria do usuário. A avaliação dos autores mostra que a estrutura proposta pode alcançar até 62% de melhoria combinada, melhorias na qualidade de serviço e no consumo energético.

O ajuste de frequência de CPU é uma estratégia de redução de consumo bastante efetiva por se ter uma relação entre a frequência e a tensão expressa pela equação $P = aCV^2F$, em que:

- $\mathbf{P}$ representa a potência dissipada pelo processador,

- $\mathbf{a}$ representa a atividade de chaveamento do processador,
- $\mathbf{C}$ representa a constante capacitiva do material,
- $\mathbf{V}$ representa a tensão de operação do processador e
- $\mathbf{F}$ representa a frequência de operação do processador.

Dessa forma, há uma relação direta de proporcionalidade entre a frequência de operação e a potência instantânea dissipada pelo processador. No entanto, através de estudos anteriores, constatamos a necessidade de se considerar o contexto de uso, pois, de outro modo, nem sempre obtém-se a redução do consumo esperada.

ÇIÇEK; GÖREN(2021) propuseram uma estratégia de personalização do gerenciamento de consumo do smartphone através do aprendizado do padrão de uso do usuário utilizando uma rede neural Convolutional Long Short-Term Memory (ConvLSTM) para aprender as dependências sequenciais baseadas no uso de bateria através de dados de séries temporais.

Os autores desenvolveram uma aplicação que coleta a cada 30 minutos e transfere para um servidor os dados de uso de bateria, para que estes dados sejam processados por um modelo ConvLSTM. O modelo proposto pelos autores prevê a capacidade restante da bateria. Os autores consideram o nível atual da bateria, a capacidade restante da bateria, a corrente instantânea, o nível de tensão e o status da bateria.

As estratégias baseadas em séries temporais para predição do consumo têm sido bastante utilizadas na área de pesquisa de redução do consumo energético. No entanto, há que se considerar dois importantes fatores:

- Quando não se considera fatores exógenos à série, a capacidade preditiva do modelo é prejudicada.
- A ausência de esclarecimentos em relação à forma como os dados da série são utilizados para treinamento e predição do modelo, uma vez que muitas pessoas trabalham com predição de séries considerando n passos à frente e utilizando os próprios dados de teste para retroalimentar o modelo, ao invés de considerar os dados de predição do modelo.

3.4 CONSIDERAÇÕES FINAIS

Trabalhos anteriores usaram estratégias de Engenharia de Software para modelar o consumo energético, como discutido em (ROMANSKY et al., 2017),(NUCCI et al., 2017), (HAO et al., 2013)

e (PATHAK et al., 2011).

Outros trabalhos utilizaram estratégias inteligentes para categorizar as aplicações pelos seus usos energéticos, como discutido em (MEHROTRA et al., 2021), e modelar o consumo energético com limitações de hardware, como discutido em (ALAWNAH; SAGAHYROON, 2017).

Em alguns trabalhos são utilizadas estratégias inteligentes para prever o uso de aplicativos por parte dos usuários, como discutido em (XIE; ZHENG; ZHANG, 2017),(DAI; HO; RUDZICZ, 2015),(BETTINI; CIVITARESE; PRESOTTO, 2020), (SARKER; ABUSHARK; KHAN, 2020) e (XIA et al., 2020).

Outros pesquisadores buscam entender a relação entre o uso de aplicativos e o respectivo consumo energético, como discutido em (DING; WANG; WANG, 2021).

Outros trabalhos usaram estratégias de aprendizado de máquina para modelar e melhorar o consumo de energia do smartphone, conforme discutido em (TARKOMA et al., 2014), (ATHUKORALA et al., 2014), (LI; LIU; MEI, 2018), (YAN; TAN; FU, 2019) e (ÇIÇEK; GÖREN, 2021).

No entanto, observou-se, a partir do levantamento realizado, uma ausência de trabalhos que visassem coletar dados de uso dos usuários de uma perspectiva de métricas do próprio sistema operacional Android, como uso de CPU, temperatura de CPU, intensidade de Sinal Wi-Fi e 3G/4G, brilho de tela, entre outros.

Coletar essas métricas que refletem o uso dos dispositivos que compõem o smartphone é importante no intuito de estabelecer a relação entre o padrão de comportamento do usuário e o uso desses dispositivos analisados.

Através dessas análises, é possível avaliar técnicas de ajuste de frequências de CPU, brilho de tela adaptativo de acordo com a intensidade das cores primárias (Vermelho, Verde, Azul), entre outras possibilidades.

Após a análise dos trabalhos anteriormente comentados, pudemos verificar algumas características que eles possuem em comum com nossa pesquisa. Na Tabela 2 elencamos essas características que os trabalhos analisados possuem em comum com nossa pesquisa.

Tabela 2 – Comparação entre os trabalhos relacionados considerados mais relevantes

Autores	Padrão de uso de App	Padrão de uso do Usuário	Modelagem de Consumo	Modelagem de uso dos dispositivos	Context Aware
(ROMANSKY et al., 2017)			✓		
(NUCCI et al., 2017)			✓		
(HAO et al., 2013)			✓		
(PATHAK et al., 2011)			✓		
(MEHROTRA et al., 2021)	✓		✓		
(ALAWNAH; SAGAHYROON, 2017)	✓		✓		
(XIE; ZHENG; ZHANG, 2017)	✓	✓			✓
(DAI; HO; RUDZICZ, 2015)	✓	✓			✓
(BETTINI; CIVITARESE; PRESOTTO, 2020)	✓	✓			✓
(SARKER; ABUSHARK; KHAN, 2020)	✓	✓			✓
(XIA et al., 2020)	✓	✓			✓
(DING; WANG; WANG, 2021)	✓	✓	✓		✓
(TARKOMA et al., 2014)	✓	✓	✓		✓
(ATHUKORALA et al., 2014)	✓	✓	✓		✓
(LI; LIU; MEI, 2018)	✓	✓	✓		✓
(YAN; TAN; FU, 2019)	✓	✓	✓		✓
(ÇIÇEK; GÖREN, 2021)	✓	✓	✓		✓
Esta Pesquisa	✓	✓	✓	✓	✓

Fonte: O autor (2022)

4 *WORKFLOW* DA SOLUÇÃO

Este capítulo apresenta uma visão geral do *workflow* proposto como uma das contribuições desta pesquisa.

O *workflow* mostrado na Figura 3 nos serviu para termos um entendimento mais aprofundado a respeito dos desafios que enfrentaríamos para modelar o uso dos dispositivos que compõem os smartphones e a relação existente entre esse uso de dispositivos e o padrão de comportamento do usuário.

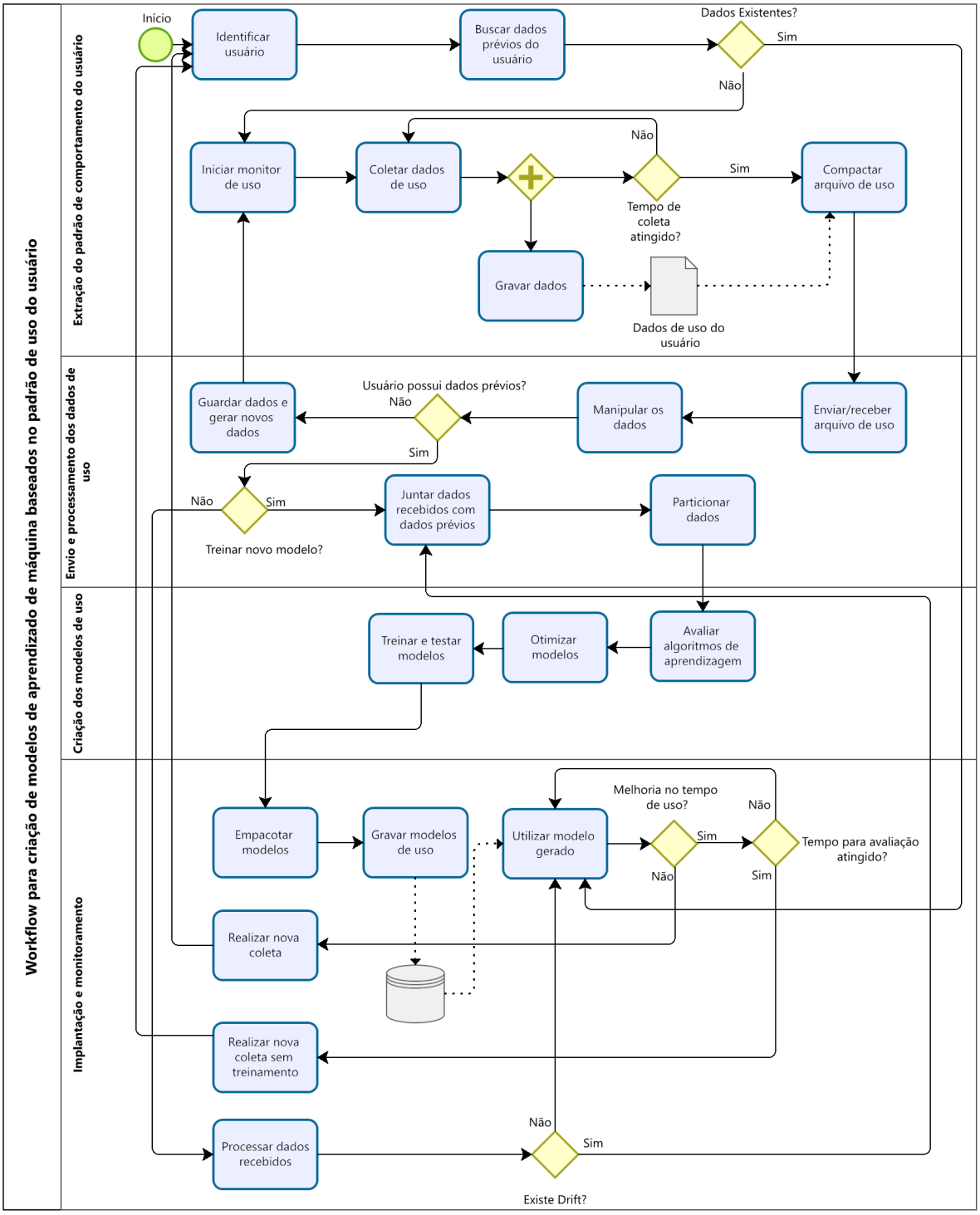
Para modelarmos o *workflow* mostrado na Figura 3, utilizamos a notação de modelagem de processos. No processo denominado **Workflow para criação de modelos de aprendizagem de máquina baseados no padrão de uso do usuário** nós identificamos quatro participantes que estabelecem as etapas necessárias para desenvolvermos a solução proposta na pesquisa descrita nesta tese.

Os participantes mostrados na modelagem do processo do *workflow* discutido neste capítulo serão posteriormente tranformados em módulos de software a fim de automatizar a execução das tarefas mostradas para cada um dos participantes. Dessa forma, temos que as tarefas do participante denominado *Extração do padrão de comportamento do usuário* serão automatizadas pelo **aplicativo de monitoramento de uso**.

Já as tarefas do participante *Envio e processamento dos dados de uso* serão automatizadas por um módulo de *software* denominado **módulo de processamento dos dados de uso**. As tarefas do participante intitulado *Criação dos modelos de uso* serão automatizadas pelo **módulo de criação dos modelos de uso** e as tarefas do participante nomeado *Implantação e monitoramento* serão automatizadas pelo **módulo de implantação e monitoramento**.

Durante a criação do processo intitulado ***Workflow* para a criação de modelos de aprendizagem de máquina baseados no padrão de uso do usuário**, adotamos a notação de **modelos** conforme discutido por BROWNLEE(2016). De acordo com BROWNLEE(2016) **modelo** é o resultado do processo de treinamento de um algoritmo de aprendizagem de máquina sobre os dados de uso do usuário utilizados durante o treinamento desses algoritmos. Iremos utilizar essa definição de modelos durante todo este trabalho. A seguir descreveremos as tarefas de cada um dos participantes modelados no processo do *workflow* mostrado na Figura 3.

Figura 3 – Workflow para criação de modelos de aprendizagem de máquina baseados no padrão de uso do usuário



Fonte: O Autor (2022)

4.1 EXTRAÇÃO DO PADRÃO DE COMPORTAMENTO DO USUÁRIO

Esse participante do processo modelado é representado na solução proposta pelo **aplicativo de monitoramento de uso** e é o responsável por identificar o usuário junto ao servidor responsável por criar os modelos de aprendizagem de máquina. Caso o **aplicativo de monitoramento de uso** não encontre dados relativos ao usuário, ele irá coletar dados de uso dos dispositivos que compõem o smartphone de modo a extrair o padrão de comportamento do usuário. A seguir descreveremos cada tarefa executada pelo participante responsável pela extração do padrão de comportamento do usuário.

Identificar o usuário: Essa tarefa realizada pelo **aplicativo de monitoramento de uso** é a responsável por identificar o usuário usando o identificador único presente em todos os aparelhos de telefonia móvel, o International Mobile Equipment Identity (IMEI). Embora pudéssemos considerar o uso de outro identificador mais associado ao usuário e não ao smartphone, verificamos em nossas análises que cada smartphone possui um conjunto de dispositivos próprios que, por sua vez, possuem um conjunto próprio de estados de uso. Dessa forma, usar modelos de aprendizagem de máquina criados previamente para estimar o uso dos dispositivos de um smartphone em um outro smartphone não seria uma decisão acertada.

Buscar dados prévios do usuário: Após a identificação do usuário realizada pela tarefa anterior, o **aplicativo de monitoramento de uso** realiza uma consulta ao servidor responsável por gerar os modelos de aprendizagem de máquina que extraem o padrão de uso do usuário. Baseada na resposta do servidor, o **aplicativo de monitoramento de uso** poderá realizar uma das seguintes ações:

- Caso ele encontre dados do usuário no servidor de modelos de uso, ele notifica o **módulo de implantação e monitoramento** para utilizar o modelo de predição de uso do usuário identificado.
- Caso contrário, o **aplicativo de monitoramento de uso** prontamente inicia uma nova coleta dos dados de uso do usuário.

Iniciar monitor de uso: Essa tarefa é responsável por inicializar os componentes de software responsáveis por monitorar o uso dos dispositivos que compõem o smartphone a fim de coletar as informações a respeito do uso do usuário.

Coletar dados de uso: Essa tarefa é responsável por coordenar e estruturar a captura das informações trazidas pelos componentes do **aplicativo de monitoramento de uso** responsáveis pelo monitoramento. Ao final da estruturação das informações trazidas pelos monitores de uso, o **aplicativo de monitoramento de uso** grava os dados e verifica se o tempo limite de coleta foi atingido, podendo, então, realizar uma das seguintes ações:

- Caso o tempo de coleta determinado seja atingido, o **aplicativo de monitoramento de uso** compacta o arquivo de uso.
- Caso contrário, o **aplicativo de monitoramento de uso** continua a realizar a coleta e a estruturação dos dados a serem gravados.

Gravar Dados: Ao final de cada ciclo de coleta e estruturação das informações de uso do usuário, o **aplicativo de monitoramento de uso** grava as informações estruturadas em um arquivo de dados de uso. Esse arquivo gerado será futuramente enviado ao servidor responsável pela geração de modelos de aprendizagem de máquina para estimar o uso do usuário.

Compactar arquivo de uso: Essa tarefa tem por finalidade a compactação do arquivo de uso do usuário no intuito de minimizar a quantidade de dados trafegados entre o smartphone e o servidor responsável por gerar os modelos de aprendizagem de máquina para estimar o uso dos dispositivos do smartphone.

Com as tarefas descritas acima, conseguimos modelar as principais atividades a serem realizadas pelo participante responsável pela Extração do padrão de comportamento do usuário, representado na Figura 3 como um dos participantes do processo. Na próxima seção vamos descrever as tarefas a serem realizadas pelo **módulo de processamento dos dados de uso**.

4.2 ENVIO E PROCESSAMENTO DOS DADOS DE USO

Esse participante do processo modelado tem como responsabilidade receber os dados produzidos pelo **aplicativo de monitoramento de uso** que executa no smartphone do usuário e realizar o processamento dos dados recebidos. Para cumprir esse objetivo, o **módulo de processamento dos dados de uso**, aqui referenciado como um participante do *workflow* modelado na Figura 3, realiza as seguintes tarefas:

Enviar/receber arquivo de uso: Esta tarefa tem por objetivo realizar a comunicação com o **aplicativo de monitoramento de uso**. Dessa forma, esta tarefa tem por objetivo receber

e descompactar os dados de uso do usuário no servidor responsável pela criação dos modelos de aprendizagem de máquina para que possam ser processados.

Manipular dados: Essa tarefa é de suma importância na execução do *workflow* descrito neste capítulo. Ela é a responsável por verificar e realizar as transformações necessárias nos dados de uso recebidos de modo que eles possam ser usados para a criação dos modelos de aprendizagem de máquina. Após manipular os dados, o **módulo de processamento dos dados de uso** verifica se o usuário possui dados previamente coletados e adota uma das seguintes ações:

- Caso o usuário possua dados previamente coletados, o **módulo de processamento dos dados de uso** realiza uma nova verificação a fim de decidir se um novo modelo de aprendizagem de máquina precisa ser treinado, tomando uma das ações:
  - Se houver a necessidade de treinar um novo modelo de aprendizagem de máquina, o **módulo de processamento dos dados de uso** repassa-os à tarefa responsável por juntar os dados recém recebidos com os dados previamente coletados do usuário.
  - Caso não haja a necessidade de treinar um novo modelo de aprendizagem de máquina, o **módulo de processamento dos dados de uso** repassa os dados recém recebidos para que possam ser processados pelo **módulo de implantação e monitoramento**.
- Caso contrário, o **módulo de processamento dos dados de uso** guarda os dados gerados e comunica o **aplicativo de monitoramento de uso** para iniciar uma nova coleta.

Guardar dados e gerar novos dados: Esta tarefa tem por objetivo guardar os dados recém manipulados e notificar o **aplicativo de monitoramento de uso** para iniciar uma nova coleta, caso o usuário não tenha nenhum dado previamente coletado. Essa ação se faz necessária porque nesta pesquisa utilizamos o paradigma de aprendizagem supervisionada para criar os modelos de aprendizagem de máquina baseados no padrão de comportamento do usuário. Segundo GÉRON(2019), na aprendizagem supervisionada, o conjunto de dados dados que você fornece ao algoritmo inclui as soluções desejadas. Dessa forma, precisamos de dados históricos para criar e avaliar os modelos de aprendizagem de máquina criados.

Juntar dados recebidos com dados prévios: Como dito anteriormente, dada a necessidade de dados históricos para criar os modelos de aprendizagem de máquina utilizando o paradigma de aprendizagem supervisionada, essa tarefa tem por objetivo juntar os dados de uso previamente existentes com os dados recém processados a fim de criar um conjunto de dados de uso com uma quantidade de exemplos de uso maior.

Particionar dados: Para realizarmos a modelagem dessa tarefa, tivemos de recorrer à literatura de aprendizagem de máquina a fim de verificarmos a melhor maneira de particionar os dados para fornecê-los aos algoritmos de aprendizagem, conforme comentamos a seguir.

De acordo com GÉRON(2019),uma das formas de categorizar os sistemas de aprendizagem de máquina é por meio da generalização. Segundo o autor, um bom desempenho dos dados de treinamento é ótimo, mas o verdadeiro objetivo é ter um bom desempenho em instâncias novas (poder de generalização). A única forma de mensurar até que ponto um modelo generalizará bem é testá-lo na prática. Para tal, GÉRON(2019) sugere dividirmos os dados em dois conjuntos: conjunto de treinamento e conjunto de testes.

Dessa forma, treina-se o modelo de aprendizagem de máquina usando o conjunto de treinamento e avalia-se seu poder de generalização usando o conjunto de testes. GÉRON(2019) sugere utilizar 80% dos dados para o conjunto de treinamento e 20% para o conjunto de testes, mas avalia um percentual menor caso haja muitos dados.

Com as tarefas descritas acima, conseguimos elencar as principais tarefas a serem realizadas pelo **módulo de processamento dos dados de uso**. As tarefas descritas acima descrevem os passos necessários para o **módulo de processamento dos dados de uso** receber e processar os dados de uso do usuário produzidos pelo **aplicativo de monitoramento de uso**. Na próxima seção, descreveremos as tarefas a serem realizadas pelo **módulo de criação dos modelos de uso**.

4.3 CRIAÇÃO DE MODELOS DE USO

Esse participante do processo modelado tem por função realizar as tarefas relativas à criação dos modelos de aprendizagem de máquina para relacionar o padrão de comportamento do usuário à utilização de dispositivos do smartphone. Para atingir esse objetivo, o **módulo de criação de modelos de uso**, aqui representado como um participante do processo mostrado na Figura 3, realiza as seguintes tarefas:

Avaliar algoritmos de aprendizagem: Ao realizar essa tarefa, o **módulo de criação dos modelos de uso** avalia diversos algoritmos de aprendizagem de máquina para escolher aqueles que produzem os modelos mais adequados para relacionar o padrão de comportamento do usuário à utilização dos dispositivos que compõem o smartphone. Para realizar essa avaliação, o **módulo de criação dos modelos de uso** utiliza os dados de treinamento e teste fornecidos pelo **módulo de processamento dos dados de uso** para treinar e testar os modelos inteligentes criados.

Otimizar modelos: Conforme comentado na Seção 2.4, uma das etapas importantes que precedem a criação de modelos de aprendizagem de máquina é o ajuste de hiperparâmetros dos algoritmos de aprendizagem de máquina. Dessa forma, esta tarefa tem por objetivo realizar a otimização de hiperparâmetros a fim de obter o modelo inteligente mais adequado que possa extrair a relação entre o padrão de comportamento do usuário e o uso dos dispositivos que compõem o smartphone. Para atingir esse objetivo, buscamos na literatura o procedimento mais adequado para realizar a otimização de hiperparâmetros dos modelos gerados, conforme descrito a seguir.

De acordo com GÉRON(2019), para evitarmos a perda de generalização do modelo gerado quando fazemos a otimização de hiperparâmetros, costuma-se utilizar um outro conjunto de dados denominado **conjunto de validação**, a fim de avaliar os modelos gerados durante o processo de otimização de hiperparâmetros. Dessa forma, particionamos o conjunto de treinamento recebido em dois subconjuntos: **conjunto de treinamento** e **conjunto de validação**, a fim de podermos executar a otimização de hiperparâmetros aderindo às melhores práticas discutidas na literatura.

Treinar e testar modelos: Ao final do procedimento de otimização de hiperparâmetros realizado na tarefa anterior, o treinamento dos algoritmos previamente elencados – cada um com seus respectivos conjuntos de hiperparâmetros mais adequados – é realizado nessa tarefa em todo o conjunto de treinamento fornecido. Após o treinamento dos respectivos modelos, cada um deles tem os seus respectivos erros de generalização mensurados no conjunto de testes previamente fornecido de forma que possam ser posteriormente empacotados e disponibilizados pelo **módulo de implantação e monitoramento**.

Ao final da realização das tarefas descritas acima, temos as principais tarefas a serem realizadas pelo **módulo de criação dos modelos de uso** elencados. Na próxima seção, descreveremos as tarefas a serem realizadas pelo **módulo de implantação e monitoramento**.

4.4 IMPLANTAÇÃO E MONITORAMENTO

No processo que descreve o *workflow* para a criação de modelos de aprendizagem de máquina baseados no padrão de comportamento do usuário, o partcipante responsável por implantar e monitorar os modelos criados corrobora de forma bastante importante para atingirmos o nosso objetivo.

No processo descrito na Figura 3, ele é o responsável por empacotar os modelos, gravar os modelos em uma base de dados que possa ser consultada, utilizar os modelos gerados para fazer predições de uso dos dispositivos que compõem o smartphone e monitorar a validade dos modelos gerados. A seguir descrevemos cada uma das tarefas executadas pelo **módulo de implantação e monitoramento** modelado na Figura 3.

Empacotar modelos: Para que os modelos possam ser disponibilizados pelo servidor de criação dos modelos de aprendizagem de máquina para os respectivos usuários, faz-se necessário gerar um arquivo no qual os modelos criados possam ser gravados. Essa tarefa tem por objetivo gerar esse arquivo.

Gravar modelos de uso: Após o empacotamento dos modelos de uso feito na etapa anterior, o **módulo de implantação e monitoramento** gera um registro em um banco de dados com a identificação do usuário e o respectivo arquivo de modelos de uso.

Utilizar o modelo gerado: Através da modelagem do *workflow* mostrada na Figura 3, percebemos que essa tarefa é executada continuamente após a geração dos modelos de uso dos dispositivos que compõem o smartphone. Durante a utilização dos modelos gerados, o **módulo de implantação e monitoramento** verifica se houve uma melhoria no uso do smartphone por parte do usuário e adota uma das seguintes ações:

- Caso esteja havendo uma melhoria no uso dos dispositivos que compõem o smartphone, o **módulo de implantação e monitoramento** verifica se o tempo de avaliação foi atingido, usando um determinado modelo inteligente de uso de dispositivo, e realiza uma das ações:
  - Caso tenha sido atingido o tempo limite para utilização de um determinado modelo de uso de dispositivo, o **módulo de implantação e monitoramento** executa a tarefa responsável por solicitar uma nova coleta de dados sem a necessidade de treinar um novo modelo de uso de dispositivo, uma vez que o modelo atual

ainda continua gerando resultados que permitam uma melhoria na utilização dos dispositivos que compõem o smartphone.

  - Caso contrário, o **módulo de implantação e monitoramento** continua utilizando o mesmo modelo.

- Caso não esteja havendo uma melhoria na utilização dos dispositivos que compõem o smartphone devido à existência dos modelos criados, o **módulo de implantação e monitoramento** responsável pelo uso dos modelos solicita uma nova coleta de dados, executanto a tarefa **Realizar uma nova coleta**, em que novos dados serão coletados e novos modelos serão gerados.

Realizar nova coleta: Ao realizar essa tarefa, o **módulo de implantação e monitoramento** notifica o **aplicativo de monitoramento de uso** para realizar uma nova coleta de dados. Ao final dessa coleta, os dados coletados seguem o fluxo para treinamento de um novo modelo de aprendizagem de máquina usando os dados preexistentes do usuário e os novos dados recém coletados.

Realizar nova coleta sem treinamento: Ao realizar essa tarefa, o **módulo de implantação e monitoramento** notifica o **aplicativo de monitoramento de uso** para realizar uma nova coleta de dados. Ao final dessa coleta, os dados são processados pelo **módulo de processamento dos dados de uso** mas não são juntados aos dados antigos nem particionados para que sejam gerados novos modelos de uso de dispositivos.

Processar dados recebidos: Ao receber os novos dados coletados e processados pelo **módulo de processamento dos dados de uso**, o **módulo de implantação e monitoramento** processa os dados recém coletados e processados para avaliar se o padrão de uso do usuário continua condizente com os dados previamente coletados e utilizados para criar o atuais modelos de uso de dispositivos que estão sendo usados pelo **módulo de implantação e monitoramento**. Para realizar essa avaliação, o **módulo de implantação e monitoramento** utiliza os conceitos de *drift* de dados e conceito, conforme discutido na Seção 2.7. Durante essa avaliação o **módulo de implantação e monitoramento** pode realizar uma das seguintes ações:

- Caso ele detecte a presença de *drift* de conceito ou de dados, o **módulo de implantação e monitoramento** notifica o **módulo de processamento dos dados de uso**. O

módulo de processamento dos dados de uso junta os dados recém coletados aos dados prévios do usuário para gerar novos modelos de aprendizagem de máquina.

- Caso contrário, o **módulo de implantação e monitoramento** continua a usar o modelo previamente criado, uma vez que o padrão de comportamento do usuário continua sendo bem modelado pelos modelos inteligentes previamente gerados.

Com as tarefas descritas acima, conseguimos modelar as principais tarefas a serem realizadas pelo participante intitulado Implantação e monitoramento. Esse participante possui um papel extremamente importante para a utilização dos modelos de aprendizagem de máquina presentes na solução proposta nesta pesquisa.

4.5 CONSIDERAÇÕES FINAIS

Após modelar o *workflow* proposto como uma das constribuições desta pesquisa e expor as tarefas realizadas por cada um dos participantes que corroboram com a execução do processo modelado, nós pudemos ter um entendimento mais aprofundado da solução proposta pela pesquisa relatada nesta tese.

Ao modelar o processo mostrado na Figura 3, pudemos compreender melhor a contribuição de cada um dos participantes do processo e como se dará a automatização das tarefas executadas nesse processo de modo a gerar o *workflow* discutido neste capítulo.

5 PROCEDIMENTO METODOLÓGICO

Este capítulo apresenta o procedimento metodológico utilizado durante a realização desta pesquisa. De acordo com WAZLAWICK(2022), o procedimento metodológico consiste na sequência de passos necessários para demonstrar que o objetivo proposto foi atingido.

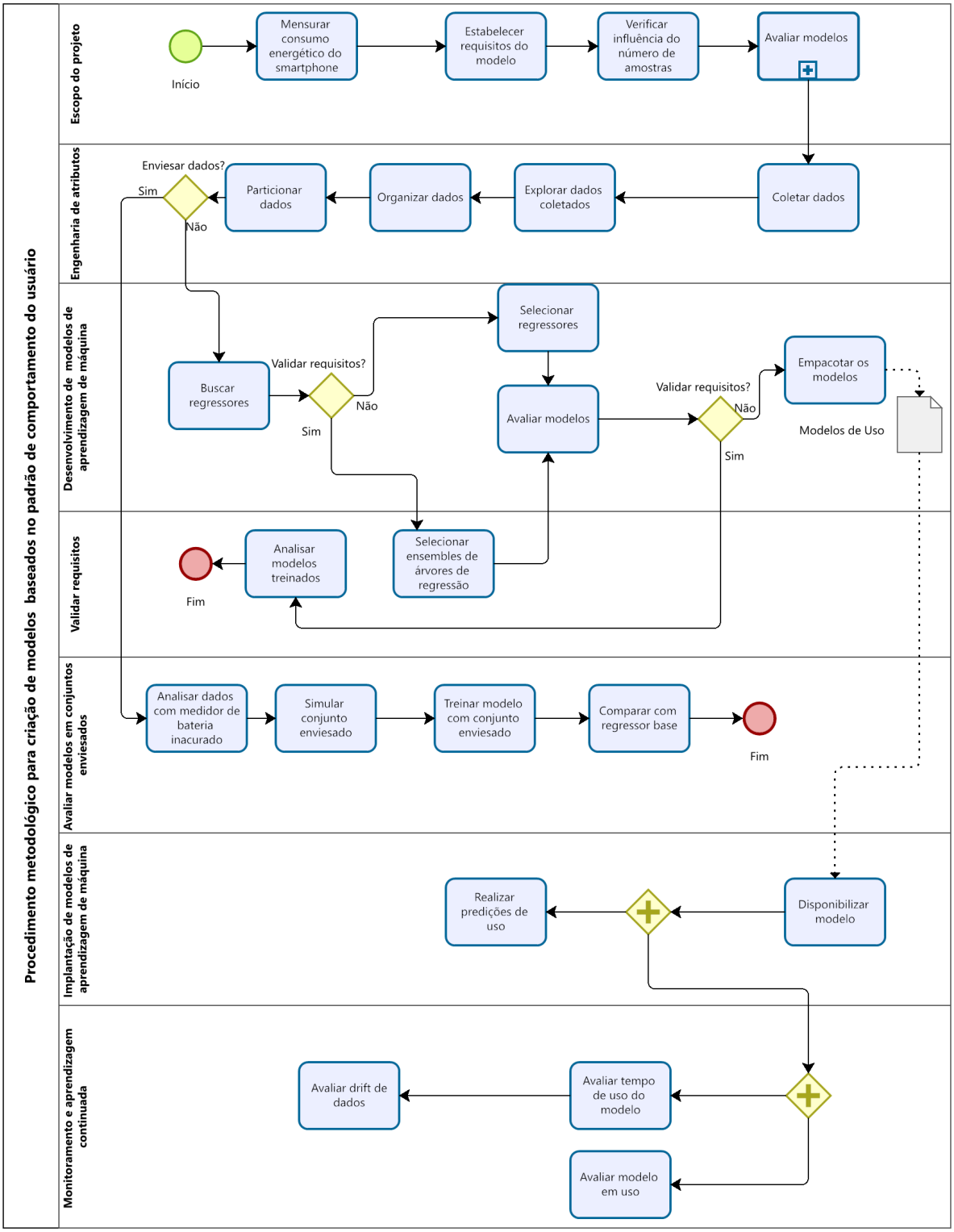
Para construir os módulos de software mostrados como participantes do *workflow* descrito no Capítulo 4, utilizamos os conceitos de engenharia de aprendizagem de máquina discutida por BURKOV(2020). De acordo com BURKOV (2020), engenharia de aprendizagem de máquina é o uso de princípios científicos, ferramentas e técnicas de aprendizagem de máquina e de engenharia de software tradicional para projetar e construir sistemas complexos de computação. Esse campo do conhecimento inclui todos os estágios, desde a coleta de dados e o treinamento de modelos até a disponibilização para uso por um produto ou os consumidores. Em outras palavras, engenharia de aprendizagem de máquina inclui qualquer atividade que permite algoritmos de aprendizagem de máquina serem implementados como parte de um sistema de produção efetivo.

De forma a uniformizar as representações, resolvemos representar o procedimento metodológico adotado nesta pesquisa utilizando a mesma notação adotada para a construção do *workflow* discutido no Capítulo 4.

Para a construção do procedimento metodológico descrito neste capítulo, utilizamos os conceitos de **modelos** e **regressores**. Conforme descrito no Capítulo 4, **modelo** é o resultado do processo de treinamento de um algoritmo de aprendizagem de máquina sobre os dados de uso do usuário utilizados durante o treinamento desses algoritmos. Como discutido na Seção 2.3, a regressão tem por objetivo prever um valor numérico como variável resposta do modelo. Dessa forma, **regressores** são modelos que possuem como variável resposta um valor numérico.

A modelagem do procedimento metodológico adotado durante esta pesquisa com a notação de modelagem de processos é mostada na Figura 4. Na Figura 4, os participantes do processo intitulado **Procedimento metodológico para criação de modelos baseados no padrão de comportamento do usuário** são os procedimentos metodológicos adotados para a construção do **aplicativo de monitoramento de uso** e dos módulos de software descritos anteriormente, no Capítulo 4. A seguir descreveremos cada uma das etapas do procedimento metodológico mostrado na Figura 4.

Figura 4 – Procedimento metodológico para criação de modelos baseados no padrão de comportamento do usuário



Fonte: O Autor (2022)

5.1 ESCOPO DO PROJETO

De acordo com HUYEN(2022), antes de desenvolvermos um sistema de aprendizagem de máquina, nós devemos entender o porquê de esse sistema ser necessário e que ele deve ser dirigido para atender aos objetivos de negócio que, por sua vez, devem ser traduzidos para objetivos de aprendizagem de máquina a fim de guiar o desenvolvimento dos modelos. Ainda segundo HUYEN(2022), antes de usarmos algoritmos de aprendizagem de máquina para resolvermos os nossos problemas, devemos enquadrar o problema a ser resolvido em uma tarefa que possa ser resolvida por aprendizagem de máquina.

Como os modelos são o resultado do treinamento de algoritmos de aprendizagem de máquina utilizando os dados de uso do usuário, precisávamos estabelcer procedimentos para coletar os dados de uso que serviriam para treinar os modelos e para avaliar esses modelos para escolhê-los da melhor forma, de modo que eles pudessem estabelecer a relação entre o padrão de comportamento do usuário e a utilização dos dispositivos que compõem o smartphone.

Esta pesquisa foi realizada utilizando smartphones que utilizam o sistema operacional Android. O Android é um sistema operacional construído usando o *kernel* do Linux adaptado. Dessa forma, para extraírmos informações a respeito dos dispositivos que compõem o smartphone, como estabelecido nos objetivos desta pesquisa discutidos na Seção 1.2, precisávamos entender o conceito de dispositivos do *kernel* do Linux.

Segundo TEACHING(2022), o modelo de dispositivo Linux foi desenvolvido para manter estruturas de dados que reflitam o estado e a estrutura do sistema. Ainda segundo TEACHING(2022), o *kernel* provê uma representação do seu modelo de dispositivos no espaço do usuário através do sistema de arquivos virtuais *sysfs* que é usualmente disponibilizado através da estrutura de diretórios */sys*. Com isso em mente, elencamos as tarefas descritas a seguir.

5.1.1 Mensurar consumo energético do smartphone

De acordo com TEACHING(2022), o subsistema de gerenciamento de energia é um dos subdiretórios do diretório */sys* no qual o *kernel* disponibiliza uma representação do seu modelo de dispositivos no espaço do usuário. Dessa forma, precisávamos estabelecer um procedimento para obter o uso de bateria, dado pelo consumo energético, do smartphone a fim de estabelecer uma relação entre o seu uso e o padrão de comportamento do usuário.

Para mensurar o consumo energético do smartphone em tempo real, utilizamos o conceito

discutido na Seção 2.1, no qual o consumo energético no tempo **t** é definido pela integral da potência instantânea no tempo **t**, como mostrado na Equação 2.2. Para obter a potência instantânea no tempo **t**, nós utilizamos a propriedade expressa pela Equação 2.1 em que a potência no instante **t** é obtida através do produto entre a tensão no instante **t** e a corrente no instante **t**.

Utilizando os conceitos discutidos na Seção 2.1 e as relações mostradas nas Equações 2.1 e 2.2, precisávamos buscar no Android ou no *kernel* do Linux, no qual o Android é baseado, uma maneira de extrair a tensão e a corrente instantâneas. Durante nossas pesquisas, nós percebemos que a Application Programming Interface (API) denominada *Battery Manager* do Android fornecia a tensão e a corrente instantâneas em Volts e em microàmpere, respectivamente(DEVELOPERS, 2020). Segundo SHEARER(2011), a seguinte relação é válida: $Joule = \frac{Watt}{Segundo}$. Dessa forma, ao projetarmos nossa solução levamos em consideração estabelecer como frequência de amostragem o intervalo de 1 segundo para que pudéssemos ter uma relação mais direta entre a potência instantânea e a energia consumida.

Devido ao nosso requisito da frequência de amostragem, antes de desenvolvermos o **aplicativo de monitoramento de uso**, fizemos testes para verificarmos se a API *Battery Manager* provida pelo Android nos fornecia informações de corrente atualizadas nesssa frequência. Ao final dos testes, verificamos que a informação de corrente instantânea provida pela API do Android não possuía uma frequência de atualização tão alta quanto desejávamos.

Diante dessa constatação, sabendo que a API provida pelo Android buscava essa infomação no subsistema de gerenciamento de energia do *kernel* do Linux, nós investigamos como obter a informação da corrente instantânea em microàmpere diretamente desse subsistema. Após investigações no subsistema de gerenciamento de energia, vimos que a informação desejada era provida pelo arquivo */sys/class/power_supply/battery/batt_current_ua_now*.

Após realizar esse estudo, temos, então, condições de coletar informações a respeito do uso da bateria do smartphone através do consumo energético, uma vez que não pudemos obter o consumo energético diretamente. Na próxima seção, discutiremos o método adotado para estabelecer os requisitos do modelo a ser criado.

5.1.2 Estabelecer requisitos do modelo

Embora esta pesquisa tenha por objetivo estabelecer uma relação entre o padrão de comportamento do usuário e o uso dos dispositivos que compõem o smartphone de forma geral,

a motivação desta pesquisa – conforme discutido na Seção 1.1– adveio da necessidade de buscar formas de relacionar o padrão de comportamento do usuário, que é a principal carga de trabalho do smartphone, com o consumo energético do smartphone. Dessa forma, para estabelecer os atributos dos dipositivos a serem monitorados, buscamos atributos dos dispositivos que estivessem relacionados ao aumento de consumo energético do smartphone.

O PROJECT(2022a) fornece um arquivo modelo para que os fabricantes possam preencher com os perfis de potência dos dispositivos que compõem os respectivos smartphones produzidos por eles, conforme pode ser visto no Apêndice A.

Para permitir que nossa solução possa coletar dados do padrão de comportamento dos usuários em qualquer smartphone sem a necessidade de recompilar o Android utilizado – estabelecendo um mecanismo de coleta pouco intrusivo –, nós buscamos projetar uma ferramenta que não possuísse acesso ao estado interno dos disposistivos que compõem o smartphone no grau de granularidade mostrada pelo arquivo de perfis de potência. Dessa forma, decidimos coletar o estado dos dispositivos utilizando algumas APIs fornecidas pelo Android e algumas ferramentas existentes no Linux – sistema no qual o Android se baseia – para coletarmos as informações relevantes a fim de estabelecer a relação entre o padrão de comportamento do usuário e o consumo energético do smartphone. A seguir mostramos a lista de atributos dos dispositivos que compõem o smartphone que foram selecionados para serem monitorados e em seguida justificamos as escolhas feitas.

1 Frequência da CPU	2 Tela	3 Wi-Fi
4 Radio(3G/4G)	5 Bluetooth	6 App em Uso
7 Uso da CPU	8 Temperatura da CPU	9 Intensidade de sinal(Móvel)
10 Intensidade de sinal(Wi-Fi)	11 RX Bytes(Móvel)	12 TX Bytes(Móvel)
13 RX Bytes(Wi-Fi)	14 TX Bytes(Wi-Fi)	15 Kb Lidos Por Segundo (Disco)
16 Kb Escritos Por Segundo (Disco)	17 Kb Lidos (Disco)	18 Kb Escritos (Disco)
19 Swap In	20 Swap Out	21 Mudanças de Contexto
22 Média(Vermelho)	23 Desvio (Vermelho)	24 Média(Verde)
25 Desvio (Verde)	26 Média(Azul)	27 Desvio (Azul)

28 Brilho 29 Período do dia 30 Orientação

A frequência e o uso de CPU aparecem como atributos elencados porque, como pode ser verificado no Apêndice A, o arquivo de perfis de potência atribui uma corrente adicional para cada uma das frequências de operação e intensidade de uso da CPU. Os atributos **Tela** e **Brilho** aparecem como atributos elencados pois a tela possui correntes instantâneas distintas para cada estado da luz de fundo, como mostrado no Apêndice A.

Os atributos relativos ao estados do **Radio(3G/4G)**, do **Wi-Fi** e das redes móvel aparecem como atributos elencados porque o arquivo de perfis de potência estabelece um acréscimo de corrente instantânea para cada um deles, tanto para a intensisdade de sinal quanto para os dados enviados e recebidos, conforme pode ser verificado no Apêndice A.

O atributo relativo ao estado do **Bluetooth** foi elencado porque o arquivo de perfis de potência atribui acréscimo de corrente para cada um dos estados do dispositivo. O atributo **Temperatura da CPU** foi elencado porque, como sabemos, uma das formas de dissipação de energia é o calor. Já o atributo do **Aplicativo em Uso** pelo usuário foi elencado porque, como sabemos, cada um dos aplicativos utilizados pelo usuário faz uso de forma distinta dos dispositivos que compõem o smartphone.

Os atributos da quantidade de dados em Kilobits (Kb) lidos e escritos por segundo e no total em disco foram elencados porque essa é a forma utilizada para mensurar o uso do dispositivo de armazenamento do smartphone. Já os atributos **Período do dia** e **Orientação** – horizontal ou vertical– foram elencados porque eles nos ajudam a identificar períodos em que o usuário mais utiliza o smartphone e o tipo de uso que o usuário faz.

Os atributos **Swap In**, **Swap Out** e **Mudanças de Contexto** foram elencados porque o arquivo de perfis de potência fornecido pelo PROJECT(2022a) atribui acréscimo de corrente para trafégo de informações no barramento da memória. Já as médias e desvios das cores dos píxels foram elencados como atributos monitorados porque cada uma das cores tem uma intensidade luminosa associada, contribuindo para identificarmos a intensidade de luz da tela, atributo esse considerado pelo arquivo de perfis de potência.

Dessa forma, conseguimos estabelecer os atributos dos dispositivos que compõem o smartphone a serem monitorados para que possamos coletar o padrão de comportamento do usuário, estabelecendo uma relação entre esse padrão comportamento e o consumo energético do smartphone.

De forma a validar os requisitos do modelo descritos nessa seção, definimos um método

descrito na Seção 5.4. Na próxima seção, descrevemos o método para avaliar a quantidade de amostras necessárias para criar os modelos que serão usados para relacionar o padrão de comportamento do usuário ao uso dos dispositivos que compoõem o smartphone.

5.1.3 Verificar Influência do número de amostras

GÉRON(2019) afirma que, em um famoso artigo publicado em 2001 por dois pesquisadores da Microsoft, Michele Banko e Erick Brill, os autores mostram que algoritmos de aprendizagem de máquina bastante distintos tiveram desempenhos semelhantes em um problema complexo de desambiguação quando alimentados por uma quantidade de dados suficiente. BANKO; BRILL(2001 apud GÉRON,2019) afirmam que "esses resultados sugerem que talvez possamos reconsiderar o custo-benefício entre gastar tempo e dinheiro no desenvolvimento de algoritmos ou empregá-los no desenvolvimento de corpus.".

A afirmação feita por BANKO; BRILL(2001) não informa a quantidade que deve ser considerada "suficiente"porque, como todo problema que envolve aprendizagem de máquina, a quantidade de dados necessária para termos um modelo com uma boa capacidade de generalização irá depender da complexidade do problema.

Dessa forma, estabelecemos um método baseado em experimentações a fim de verificar a quantidade de dados necessária para que tenhamos uma estabilização do erro do modelo. Para realizar a avaliação dos modelos, precisávamos estabelecer a métrica a ser utilizada. Segundo GÉRON(2019), uma métrica típica para avaliar o desempenho dos modelos é a métrica RMSE, que pode ser expressa pela Equação 5.1. Na Equação 5.1, **h** representa o modelo de aprendizagem de máquina como uma função dos dados de entrada do modelo, **X** representa o conjunto de dados de entrada sobre o qual o modelo está sendo avaliado e **Y** é o conjunto de valores respostas usado para avaliar o quão próximo os valores preditos pelo modelo estão dos valores do conjunto de dados sobre o qual o modelo está sendo avaliado.

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i)^2} \tag{5.1}$$

Para avaliar a influência da quantidade de amostras na diminiução do erro do modelo, utilizaremos um regressor base. Um regressor base é construído utilizando um algoritmo de aprendizagem de máquina de baixa complexidade e com os hiperparâmetros com seus valores padrões. Em aprendizagem de máquina, utilizamos um regressor base para compará-lo a um

outro regressor que é treinado utilizando qualquer tipo de ajuste nos dados ou nos hiperparâmetros a fim de avaliar a eficácia das técnicas aplicadas na redução do erro do modelo. A seguir, listamos a sequência de passos adotados para verificar a estabilização de erros do modelo treinado.

1. Inicia a contagem de amostras para 0.
2. Utiliza um regressor base para avaliação da melhoria do erro.
3. Soma 500 ao contador atual de amostras.
4. Utiliza a quantidade de amostras indicada pelo contador de amostras para treinar o regressor base.
5. Avalia o erro cometido pelo regressor base em um outro conjunto, denominado **conjunto de testes**.
6. Repete os passos 3-5 por 35 vezes e avalia a redução na média dos erros do modelos treinados. Fizemos 35 repetições para avaliar a cada hora de experimento, por aproximadamente 5 horas, o impacto na redução do erro do regressor de base.

Ao final do método descrito, pudemos verificar como o erro dos modelos se comportam e estimar uma quantidade de amostras "suficiente"para a nossa tarefa de aprendizagem de máquina: construir modelos que sejam capazes de estabelecer a relação entre o padrão de comportamento do usuário e a utilização dos dispositivos que compõem o smartphone. Na próxima seção, descrevemos o método utilizado para avaliar os modelos a serem criados pelos módulos de *software* desenvolvidos durante a pesquisa.

5.1.4 Avaliar Modelos

HUYEN(2022) atribui o sucesso dos sistemas de aprendizagem de máquina aos dados sobre os quais esses sistemas são treinados. Por isso, fizemos uma vasta investigação para elencar os atributos dos dispositivos que compõem os smartphones a serem monitorados e sobre a quantidade de dados a serem utilizados no treinamento dos modelos criados nas seções anteriores. Outro fator crucial elencado por HUYEN(2022) é o viés indutivo que, como discutido na Seção 2.3, é necessário para restringir os modelos a serem avaliados no espaço de busca,

tornando possível a um algoritmo de aprendizagem fazer generalizações a partir de um conjunto de treinamento.

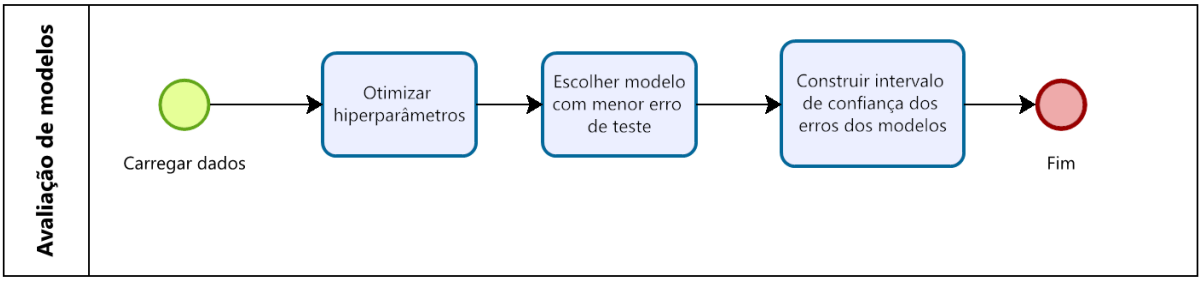
Dessa forma, se faz importante estabelecer um método a ser seguido para avaliar os modelos a serem construídos durante esta pesquisa. Para realizar a avaliação dos modelos, pelos motivos comentados na Seção 5.1.3, escolhemos a métrica RMSE.

No entato, ainda segundo GÉRON(2019), em um processo de treinamento, cujo objetivo é otimizar o conjunto de parâmetros do modelo, é mais fácil minimizar uma outra métrica – a MSE– da qual a RMSE é derivada. A métrica MSE pode ser expressa pela Equação 5.2. Na Equação 5.2, os termos **X**, **h** e **Y** têm o mesmo significado expresso pela Equação 5.1.

$$MSE(X, h) = \frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i)^2 \tag{5.2}$$

Dessa forma, utilizamos a métrica de erro MSE para avaliar os modelos gerados durante a otimização de hiperparâmetros e o treinamento de modelos. A métrica RMSE é utilizada durante o processo de avaliação de modelos para escolher o modelo com menor erro no conjunto de testes e para construir o intervalo de confiança dos erros dos modelos gerados. O método adotado para realizar a avaliação dos modelos é mostrado na Figura 5. A seguir descreveremos as tarefas a serem executadas para avaliar os modelos de aprendizagem de máquina.

Figura 5 – Avaliação de modelos



Fonte: O Autor (2022)

Carregar dados: O processo de avaliação de modelos se inicia carregando os dados sobre os quais o algoritmo de aprendizagem será treinado para gerar os modelos a serem avaliados.

Otimizar hiperparâmetros: Conforme discutido anteriormente na Seção 2.4, uma das etapas importantes que precede a criação de modelos é a otimização dos hiperparâmetros dos algoritmos de aprendizagem de máquina. Conforme discutido na Seção 4.3, costuma-se utilizar o **conjunto de validação** para a realização dessa tarefa.

Dessa forma, realizamos a seguinte sequência de passos para otimizar os hiperparâmetros dos algoritmos de aprendizagem:

1. Escolhemos um conjunto de valores para os hiperparâmetros do algoritmo de aprendizagem.
2. Treinamos o algoritmo com o conjunto de hiperparâmetros escolhidos no passo anterior utilizando o conjunto de treinamento.
3. Mensuramos o erro do modelo utilizando a métrica de erro MSE sobre o conjunto de validação.
4. Repetimos os passos 1,2 e 3 por 30 vezes.
5. Ao final, escolhemos o conjunto de hiperparâmetros para o algoritmo de aprendizagem de máquina que está sendo avaliado que produz o modelo com o menor erro no conjunto de validação.

Escolher o modelo com o menor erro de teste: Após escolhermos o conjunto de hiperparâmetros que produz o modelo com o menor erro no conjunto de validação para o algoritmo de aprendizagem de máquina que está sendo avaliado, treinamos o algoritmo em questão por 30 vezes, gerando 30 modelos distintos. Avaliamos os 30 modelos previamente gerados usando o nosso conjunto de testes e escolhemos o modelo que possua o menor erro no conjunto de testes. Para realizar essa avaliação utilizamos a métrica de erro RMSE.

Construir intervalo de confiança dos erros dos modelos: Como forma de avaliar a estabilidade dos modelos gerados com o conjunto de hiperparâmetros previamente escolhidos, nós construímos um intervalo de confiança para a média dos erros dos 30 modelos gerados no passo anterior, avaliados usando a métrica RMSE.

Para gerarmos esse intervalo de confiança para a média dos erros dos 30 modelos gerados, nós utilizamos o conceito de *Bootstrapping*. Segundo EFRON; TIBSHIRANI(1994), para uma boa estimativa dos limites de confiança seriam necessárias mais de 500 replicações. Baseados nessa afirmação, durante nossas análises para construção do intervalo de confiança, fizemos 1000 replicações das médias dos erros dos 30 modelos gerados previamente.

Ao longo desta pesquisa, para estabelecermos se há uma diferença significativa entre os modelos, consideramos sempre dividir o intervalo inter-quartil pela mediana dos dados sobre o qual os modelos gerados estão sendo analisados a fim de obtermos um percentual de variação

dos dados. Utilizando essa informação, consideramos o erro cometido pelo modelo com menor erro entre os melhores modelos elencados e verificamos quais modelos possuem um erro cuja diferença percentual seja maior que a diferença percentual de variabilidade obtida anteriormente. Os modelos que possuírem um erro maior que a diferença percentual de variabilidade obtida são considerados por nós com uma diferença significativa em relação ao modelo com menor erro no conjunto de dados analisado.

Ao realizar as tarefas descritas acima para definirmos o escopo de nosso projeto, nós temos os procedimentos estabelecidos para coleta de dados e avaliação de modelos, tarefas fundamentais para que possamos alcançar os objetivos definidos na Seção 1.2. Na próxima seção, vamos definir os procedimentos necessários para realizarmos uma outra etapa fundamental na construção de modelos inteligentes: a engenharia de atributos.

5.2 ENGENHARIA DE ATRIBUTOS

De acordo com HUYEN(2022), em 2014 foi publicado um artigo intitulado "Practical Lessons from Predicting Clicks on Ads at Facebook"no qual os autores defenderam que ter os atributos corretos era o elemento mais importante no desenvolvimento de modelos.

Muitas empresas têm observado que, possuindo um modelo funcional, ter os atributos corretos tende a proporcionar a elas o maior aumento no desempenho em comparação às técnicas aplicadas aos algoritmos inteligentes, como ajuste de hiperparâmetros(HUYEN, 2022).

Dessa forma, se faz importante estabelecer um método a ser seguido para realizar a engenharia de atributos de modo a preparar os dados para a criação dos modelos. Engenharia de atributos são os procedimentos adotados para transformar os dados coletados, chamados de dados "brutos", em dados que possam ser melhor utilizados pelos modelos.

A seguir descreveremos as tarefas a serem executadas para realizar a engenharia de atributos.

Coletar Dados: Após definirmos os atributos dos dispositivos, conforme descrito na Seção 5.1.2, nós precisávamos estabelecer um procedimento para coletar os dados selecionados. O procedimento adotado para coletar informações a respeito dos atributos dos dispositivos que compõem o smartphone e desenvolver o **aplicativo de monitoramento de uso** está descrito no Apêndice B.

Explorar dados coletados: A exploração de dados ocorre quando o módulo responsável

descreve os dados para obter informações a respeito do conteúdo e da estrutura dos dados, provendo um conjunto de metadados como valor máximo, mínimo e médio. O módulo também executa funções de detecção de erros na etapa de validação dos dados, varrendo os dados em busca de erro. Para atingir o objetivo dessa tarefa, nós projetamos um módulo chamado **Processamento de dados** no qual realizamos uma análise exploratória dos dados para caracterizá-los, como mostrado na Módulo 1.

Organizar dados: No processo de engenharia de atributos, essa tarefa é responsável por reformatar alguns atributos e corrigir erros nos dados. O módulo **Processamento de dados**, como mostrado no Módulo 1, realiza essa atividade vital que prepara os dados para construir os modelos no próximo estágio do projeto.

Particionar dados: Nessa tarefa, os dados são particionados em dois conjuntos a serem usados pelos modelos de aprendizagem de máquina a serem criados: treinamento e teste. Como comentado na Seção 4.3, o **conjunto de treinamento** é usado para ajustar os parâmetros do modelo. A partir do conjunto de treinamento, geramos um outro conjunto – o **conjunto de validação** – que é utilizado durante o processo de otimização de hiperparâmetros. Já o **conjunto de testes** é usado para avaliar o poder de generalização do modelo gerado. Para gerar os dois conjuntos, nós realizamos a seguinte sequência de passos:

1. Embaralhamos os dados obtidos de modo a não favorecer nenhum algoritmo a ser criado e avaliado.

2. Dividimos os dados em dois conjuntos: **conjunto de treinamento** e **conjunto de testes**. Para realizar essa divisão, seguimos a recomendação dada por PRECHELT et al.(1994), dividindo os dados na seguinte proporção: 70% dos dados para o **conjunto de treinamento** e 30% dos dados para o **conjunto de testes**.

3. No **conjunto de treinamento** nós dividimos o conjunto tal que: 80% forme o conjunto a ser usado para treinamento do algoritmo e 20% o **conjunto de validação**, seguindo a recomendação dada por GÉRON(2019). Para separar os 20% a serem usados para o **conjunto de validação**, nós utilizamos o procedimento proposto NG et al.(1997), conforme descrito a seguir.

Para construir o conjunto de validação, nós usamos a estratégia proposta por NG et al.(1997). NG et al.(1997) que mostra resultados bastante interessantes quando selecionamos

modelos com um erro maior de validação cruzada sobre outros modelos, para evitar o *overfitting* no conjunto de validação. A estratégia proposta pelo autor é composta de três passos:

1. Gere tantos modelos quanto possível;
2. teste-os no conjunto de validação, e coloque-os em ordem decrescente e
3. para algum **k**, pegue o modelo validado no k-ésimo percentil do conjunto de validação.

Nós usamos essa abordagem usando um subconjunto de dados de validação com as instâncias em ordem aletória, como mostrado na Módulo 1. O módulo de engenharia de atributos, após o particionamento dos dados pode ser usado para a realização de duas outras tarefas:

1. Caso se deseje avaliar o erro de modelos gerados a partir de conjuntos de dados enviesados, os dados particionados são fornecidos ao módulo responsável por fazer esse tipo de avaliação.
2. Caso contrário, o módulo responsável pelo processamento dos dados fornece os dados ao módulo responsável pela criação de modelos.

A seguir, nós mostraremos o módulo que projetamos para realizar a engenharia de atributos. Em seguida, explicaremos as funcionalidades projetadas e o porquê de termos projetado elas.

Módulo 1: Engenharia de atributos

```
Global: dataset
1  Function explore(features):
2      data = dataset[features]
3      mean = mean(data)
4      std_dev = std_dev(data)
5      max = max(data)
6      min = min(data)
7      median = median(data)
8      results = dict('mean':mean, 'std_dev':std_dev, 'max':max,'min':min,
         'median':median)
9      return results
10 Function data_spliting(X,y,val_size):
11     X_train,y_train,X_val,y_val = split(X,y,val_size)
12     X_val,y_val = sample(X_val,y_val,sample_size)
13     return X_train,y_train,X_val,y_val
14 Function wrangling_data(predictors, target):
15     data = copy(dataset)
16     data['Power(mW)'] = data['Current'] × data['Volt(V)']
17     data = drop_features(data,['Current','Volt(V)'])
18     cat_cols = filter_columns(data,dtype='categorical')
19     num_cols = filter_columns(data,dtype='numerical')
20     data = transformer(data,cat_cols,OridinalEncoder)
21     data = transformer(data,num_cols,MinMaxScaler)
22     data = transformer(data,num_cols,QuantileTransformer,'normal')
23     data = remove_outliers(data)
24     X = data[predictors]
25     y= data[target]
26     return X,y
```

Fonte: Elaborado pelo autor (2022)

Na função **explore** nós provemos os atributos que queremos explorar no conjunto de dados. A função realiza a exploração e nos retorna os valores de **média**,**desvio padrão**,**valor**

mínimo, **valor máximo** e **mediana** para os atributos fornecidos.

Na função **wrangling_data**, nós provemos a lista de atributos que nós queremos usar como preditores em nosso modelo. Além disso, informamos o atributo que queremos usar como resposta do nosso modelo. Após fornecermos essas informações, a função realiza as atividades relativas à limpeza, organizando os dados corretamente e retornando os conjuntos de dados que serão utilizados pelos algoritmos de aprendizagem de máquina.

Usando abordagem descrita na Seção 5.1.1, nós fazemos a engenharia de atributos para obter a potência instantânea usando a corrente e a tensão instantâneas. Após esse procedimento, excluímos os atributos de corrente e a tensão instantâneas, uma vez que eles não melhoram a capacidade preditiva do modelo, tornando-o mais complexo.

Outras tarefas importantes que devem ser realizadas a fim de preparar os dados para os algoritmos de aprendizagem de máquina são: transformar os dados categóricos e escalonar os dados numéricos. Além disso, precisamos ajustar as distribuições dos atributos, tornando-as mais aderentes à distribuição normal e removendo os *outliers*.

Ao final da função **data_wrangling**, nós temos os dados preparados de forma adequada a serem usados para treinar os algoritmos de aprendizagem dividindo os dados em dois conjuntos: conjunto dos atributos preditores e conjunto do atributo resposta.

O Módulo 1 disponibiliza uma função chamada **data_spliting** que permite a divisão dos dados de treinamento em dois conjuntos **treinamento** e **validação**.

Ao realizar as tarefas descritas acima, nós temos os procedimentos estabelecidos para realizar a engenharia de atributos, preparando os dados para a criação dos modelos que irão relacionar o padrão de comportamento do usuário ao uso dos dispositivos que compõem o smartphone. Na próxima seção, vamos definir o método necessário para criarmos os modelos.

5.3 DESENVOLVIMENTO DE MODELOS DE APRENDIZAGEM DE MÁQUINA

HUYEN(2022) afirma que, para construir um modelo de aprendizagem de máquina, nós primeiramente temos que selecionar o modelo a ser construído. A autora também afirma que o desenvolvimento de modelos é um processo iterativo.

Baseados nessas ideias, nós definimos um procedimento para a construção dos modelos de aprendizagem de máquina a serem utilizados para estabelecer a relação entre o padrão de comportamento do usuário e a utilização dos dispositivos, conforme explicado a seguir.

Buscar regressores: Ao realizar essa tarefa, o módulo responsável avalia diversos algoritmos de aprendizagem de máquina de modo a estabelecer uma classificação de acordo com o erro gerado. Como discutido na Seção 2.3, durante a realização desta pesquisa, optamos por utilizar os modelos rasos(*shallow*) em oposição às redes neurais profundas(*deep*).

Dessa forma, para a realização dessa tarefa, limitamos a busca para avaliarmos apenas os algoritmos de aprendizagem rasos em suas configurações padrões. Para realizar a avaliação e a classificação dos algoritmos analisados, utilizamos como métrica a RMSE, conforme a Equação 5.1. Ao final da classificação dos algoritmos, essa tarefa pode realizar uma das seguintes ações:

- Caso a busca e classificação esteja sendo realizada no intuito de validar os atributos selecionados dos dispositivos que compõem o smartphone, conforme discutido na Seção 5.1.2, a tabela com a classificação dos regressores é passada ao módulo responsável por validar os atributos selecionados, conforme descrito na Seção 5.4.

- Caso contrário, a tabela gerada por essa tarefa é repassada para outra tarefa do módulo de criação dos modelos responsável pela seleção dos regressores.

Selecionar regressores: A fim de avaliar a quantidade de regressores a serem selecionados para a criação dos modelos e comparação nos próximos estágios, nós estabelecemos um método composto dos seguintes passos:

1. Verificar o erro cometido pelo regressor que está na primeira colocação da tabela fornecida pela tarefa responsável pela busca dos algoritmos.

2. Selecionar até 5 algoritmos que possuam um erro não superior a 50% do erro cometido pelo regressor que está na primeira posição da tabela. Estabelecemos um quantitativo não superior a 5 porque, durante experimentos preliminares, notamos que os modelos que assumem uma colocação inferior não melhoram com o ajuste de hiperparâmetros realizado na etapa de avaliação de modelos.

Avaliar modelos: Após selecionar os regressores na etapa anterior, realizaremos os passos descritos na Seção 5.1.4. Ao final da avaliação, o módulo responsável pela criação dos modelos executa uma das seguintes ações:

- Caso a avaliação esteja sendo feita no intuito de prover modelos para avaliar o quão os atributos selecionados dos dispositivos que compõem auxiliam na capacidade preditiva

deles, o resultado dessa tarefa é repassado para o módulo responsável por validar os atributos.

- Caso contrário, o resultado da avaliação é repassado à tarefa responsável por empacotar os modelos gerados de modo que possam ser utilizados para fazer predições.

Empacotar os modelos: Essa tarefa, a ser realizada pelo módulo responsável por gerar os modelos, tem por finalidade tornar os modelos gerados implantáveis. Nós pensamos em um formato, durante a fase de projeto, que seja fácil de transferir e gerenciar.

Para realizar as tarefas descritas acima, projetamos um módulo de *software* intitulado **Criação de modelos** – em que nós usamos os dados providos pelo módulo **Processamento de dados**–, buscamos os regressores, avaliamo-los eles e os empacotamos, como mostrado no Módulo 2.

Módulo 2: Criação de Modelos

```
1  Function fit_models(predictors,target,select_features):
2      data = dict()
3      X,y = wrangling_data(predictors,target)
4      X_train,y_train,X_val,y_val = split(X,y,val_size)
5      estimators = evaluate_algorithms(X_train,y_train,X_val,y_val)
6      best_estimators = estimators[:5]
7      for estimator in best_estimators do
8          model = optimize(X_train,y_train,X_val,y_val, estimator)
9          errors = array()
10         for i in 1..30 do
11             X_train,y_train,X_val,y_val = shuffle(X_train,y_train,X_val,y_val)
12             model.fit(X_train,y_train)
13             error = evaluate(X_val,y_val,model)
14             errors.add(error)
15         data[model.name]={'model':model,'errors':errors}
16 Function rfe_feature_selection(predictors,target,num_votes):
17     data = dict()
18     X,y = wrangling_data(predictors,target)
19     X_train,y_train,X_val,y_val = split(X,y,val_size)
20     estimators = evaluate_algorithms(X_train,y_train,X_val,y_val)
21     best_estimators = estimators[:5]
22     for estimator in best_estimators do
23         rfe = RFE(estimator,X_train,y_train)
24         selected_features = rfe.support ()
25         data[estimator.name]={'estimator':estimator, 'selected
             features':selected_features}
26     return data
```

```
1  Function selectGApredictors,target,num_votes:
2      data = dict() X_train,y_train,X_val,y_val =
         wrangling_data(predictors,target)
3      estimators = evaluate_algorithms(X_train,y_train,X_val,y_val)
4      best_estimators = estimators[:5]
5      individual_votes = array()
6      for estimator in best_estimators do
7          hall_of_fame = make_GA(estimator,X_train,y_train)
8          vote = hall_of_fame[0]
9          individual_votes.add(vote)
10     votes = sum(individual_votes)
11     selected_features = votes ≥ num_votes
12     data= {'estimators':estimators,'selected features':selected_features}
13     return data
14 Function make_GA(estimator,X,y):
15     tools = dict()
16     tools['individual creator']=individual_creator()
17     tools['mate'] =cx_two_point
18     tools['mutate']=mut_flip_bit
19     tools['evaluate']=evaluate
20     hall_of_fame=run_GA(X,y,estimator,tools)
21 Function mutual_info(model,X,y):
22     n_features = dict()
23     for i in 1..X.shape[column] do
24         X_train,y_train,X_val,y_val = shuffle(X_train,y_train,X_val,y_val)
25         X_train,X_val = select_k_best(X_train,X_val, i)
26         model.fit(X_train,y_train)
27         error = evaluate(X_val,y_val,model)
28         n_features[i]=error
29     return n_fetures.sort()[0]
```

Fonte: Elaborado pelo autor (2022)

Para projetar o nosso módulo **Criação de modelos** alguns aspectos precisaram ser considerados:

- Qual a estratégia mais apropriada a ser considerada para ajustar os hiperparâmetros dos modelos?
- Quais as estratégias a serem consideradas para realizar a seleção de atributos?

Para avaliar a técnica mais apropriada a fim de ajustar os hiperparâmetros dos algoritmos classificados pela técnica de avaliação do estágio anterior, nós investigamos *benchmarks* para comparar as técnicas mais apropriadas e escolher aquela com melhor custo-benefício e, conforme discutido na Seção 2.4, escolhemos a técnica de otimização bayesiana.

Além da otimização de hiperparâmetros, nós precisamos investigar outra tarefa: seleção de atributos. De acordo com Kuhn e Jonson (2019), a principal motivação para realizar a seleção de atributos deve ser: mitigar um problema específico entre os preditores e um modelo ou reduzir a complexidade do modelo. Então, os autores põem como principal objetivo da seleção de atributos a redução da complexidade do modelo sem comprometer a capacidade preditiva dele.

Como discutimos na Seção 2.5, há duas abordagens para se fazer seleção de atributos: gulosa e não-gulosa. Dessa forma, baseados nas ideias discutidas por Kuhn e Jonson (2019), nós projetamos uma estratégia gulosa, para seleção de atributos, usando Recursive Feature Elimination (RFE) e seleção por informação mútua, e uma estratégia não-gulosa, usando seleção de atributos por Genetic Algorithm (GA).

A escolha do RFE para realizar a seleção de atributos se deu pelo fato de ele ser um método *wrapper* que proporciona uma escolha de atributos mais adequada ao algoritmo que está sendo usado para realizar esse tipo de seleção de atributos. Já a escolha da técnica de seleção de atributos baseada em informação mútua se deu pelo fato de, conforme comentado na Seção 2.5, essa técnica possuir um custo computacional mais baixo por fazer a seleção baseada na filtragem. KUHN; JOHNSON(2019) afirmam que a forma como os algoritmos genéticos funcionam possibilita a eles aproveitar boas soluções ao longo da execução para construir melhores soluções convergindo para um *plateau* de otimização. Esse fato nos fez escolher GA como estratégia não gulosa para fazer seleção de atributos.

Realizando as tarefas acima descritas e projetando o módulo levando em consideração questões a respeito de busca de regressores, otimização de hiperparâmetros e seleção de

atributos, nós conseguimos estabelecer um método para a construção do módulo **Construção de modelos**. Na próxima seção, vamos discutir os procedimentos adotados para validar os requisitos para a construção dos modelos, discutidos na Seção 5.1.2.

5.4 VALIDAR REQUISITOS

Para validar os atributos selecionados dos dispositivos que compõem o smartphone, conforme descrito na Seção 5.1.2, precisávamos utilizar alguma técnica de interpretabilidade de modelos. Para entender a respeito de interpretabilidade de modelos, buscamos na literatura as técnicas mais utilizadas.

De acordo com KAMATH; LIU(2021), a inovação em aprendizagem de máquina trouxe grandes avanços no poder de predição e na acurácia, embora estejam cada vez mais complexos. KAMATH; LIU(2021) afirmam que esse infeliz compromisso entre uma melhor qualidade e a transparência traz sérias consequências para a confiança e adoção dos modelos.

Uma forma de melhorar a transparência dos modelos gerados é usando modelos interpretáveis. Modelos interpretáveis são os modelos que possuem propriedades que tornam compreensível a geração dos resultados produzidos por eles a partir das entradas fornecidas.

As técnicas de interpretabilidade *post-hoc* representam uma vasta coleção de métodos criados para lidar especificamente com o problema dos modelos caixa-preta, no qual nós não temos acesso à representação interna dos atributos ou à estrutura interna do modelo(KAMATH; LIU, 2021). Nós podemos dividir os métodos *post-hoc* em três categorias:

Explicações visuais: Utilizam figuras para expressar e interpretar relacionamentos entre os atributos do modelo e as predições de uma forma legível.

Importância de atributos: Permite-nos quantificar o relacionamento dos atributos preditores com a variável resposta de forma mais precisa.

Explicações baseadas em exemplos: Buscam explicar os modelos usando a predição de amostras individuais do modelo.

Baseados nas técnicas *post-hoc* de interpretabilidade de modelos, nós utilizamos a técnica de **Importância de atributos**. Um dos principais algoritmos de aprendizagem de máquina que possuem essa técnica implementada são os algoritmos baseados em árvores de regressão. Dessa forma, utilizamos modelos *ensemble* de árvores – pois possuem uma melhor capacidade de generalização quando comparados aos modelos que utilizam apenas uma árvore de regressão

– com os atributos selecionados, conforme descrito na Seção 5.1.2 – para validar se os atributos selecionados foram de fato importantes para a melhoria da capacidade de generalização desses modelos. Para realizar essa validação adotamos os seguintes passos:

Selecionar ensembles de árvores de regressão: Utilizando os dados previamente coletados e a tabela de classificação dos modelos construída pela tarefa **Buscar regressores**, nós selecionamos na tabela de classificação os *ensembles* de árvores de regressão.

Avaliar Modelos: Após a seleção dos modelos feitos na tarefa anterior, nós avaliamos os modelos selecionados utilizando os procedimentos descritos na Seção 5.1.4.

Analisar modelos treinados: Conforme explicado na Seção 5.1.4, durante o processo de avaliação, o modelo que possui o menor erro para cada algoritmo selecionado é separado dos outros 30 modelos treinados. Utilizando esses modelos selecionados, nós avaliamos a importância dos atributos selecionados.

Com esse método descrito, nós temos condições de validar se os atributos selecionados por nós, seguindo o procedimento descrito na Seção 5.1.2, de fato contribuem para a capacidade preditiva dos modelos de regressão construídos pelos módulos criados. Na próxima seção, iremos discutir o método adotado para avaliar os modelos gerados mesmo quando o conjunto de dados está enviesado.

5.5 AVALIAR MODELOS EM CONJUNTOS ENVIESADOS

Para o desenvolvimento desta pesquisa, propomos como objetivo geral a criação de modelos de uso dos dispositivos baseados no padrão de comportamento do usuário. Ao propormos uma pesquisa beaseada nesse objetivo, a qualidade dos dados a serem usados é de fundamental importância.

De acordo com GÉRON(2019), se os dados de treinamento estiverem cheios de erros, *outliers* e ruídos (devido a medições de baixa qualidade), os modelos terão mais dificuldade de encontrar os padrões básicos, prejudicando a capacidade de generalização. Diante dessa constatação, estabelecemos um método para avaliar a capacidade preditiva do modelo em conjuntos de dados enviesados, conforme descrito a seguir.

Analisar dados com medidor inacurado: Alguns smartphones não possuem um circuito medidor de bateria capaz de medir com exatidão a corrente instantânea e a respectiva potência instantânea dissipada. Esse tipo de circuito medidor de bateria que não tem a capacidade

de medir com exatidão a corrente instantânea é intitulado **circuito medidor de bateria inacurado**.

Dessa forma, para avaliar o erro do modelo gerado por conjuntos de dados enviesados, uma vez que a motivação desta pesquisa veio a partir da investigação de estratégias para a redução do consumo energético, nós decidimos obter dados de corrente e tensão usando smartphones que não possuam medidores de bateria acurados. O objetivo dessa tarefa é analisar os dados obtidos com esse tipo de circuito medidor de bateria para verificar as frequências com que o valor da corrente instantânea se repetia, mesmo com demandas distintas feitas pelo usuário no smartphone avaliado.

Simular conjunto enviesado: Como comentado anteriormente, na Seção 5.2, os dados obtidos, na tarefa responsável por particioná-los, são particionados em dois conjuntos: **conjunto de treinamento e conjunto de testes**.

Na Seção 5.2, nós dicutimos também que o **conjunto de testes** é utilizado durante a avaliação do modelo para verificar a sua capacidade de generalização. Dessa forma, para avaliar qual a perda da capacidade de generalização dos modelos criados usando a nossa estratégia, nós precisamos utilizar um conjunto de testes não enviesado.

Com isso, o objetivo dessa tarefa é, após a análise feita na tarefa anterior, replicar o comportamento do **medidor de bateria inacurado** em um conjunto de treinamento obtido a partir de dados gerados por um **medidor de bateria acurado** de forma que possamos avaliar o modelo gerado em um **conjunto de testes** não enviesado.

Treinar modelo com conjunto enviesado: Para a realização dessa tarefa, realizamos os mesmos procedimentos descritos na Seção 5.3, bucando regressores, selecionando os regressores e avaliando os modelos gerados. A única diferença em relação ao procedimento descrito na Seção 5.3 é que nessa tarefa nós treinamos o modelo com o **conjunto de treinamento** enviesado por simulação e testamos com o **conjunto de testes** original.

Como comentado anteriormente, ao treinarmos usando um conjunto de dados enviesados e testarmos o modelo gerado usando um conjunto de testes não enviesado, somos capazes de identificar a perda de generalização feita pelo enviesamento do modelo.

Caso optassemos por utilizar os dados enviesados para treinamento e teste dos modelos criados, não poderíamos avaliar a perda de generalização do modelo devido ao enviesamento. Tal fato ocorreria porque os dados utilizados para verificar o quão próximas as predições dos modelos criados estão das medições reais também estariam enviesados.

Comparar com regressor base: Como comentado anteriormente na Seção 5.1.3, um regressor base é normalmente utilizado para avaliar o quão as estratégias usadas para melhoria da capacidade preditiva de um determinado modelo são efetivas. Como um **medidor de bateria inacurado** tem o comportamento de repetir os valores de corrente instantânea medidos, nessa tarefa utilizamos como regressor base um modelo que tenha esse comportamento. Dessa forma, o regressor base utilizado para comparar o modelo enviesado criado tem o comportamento expresso pela Equação 5.3.

$$P_t = P_{t-1} \tag{5.3}$$

Na Equação 5.3 P_t representa a potência instantânea no instante atual e P_{t-1} representa a potência instantânea no instante imediatamente anterior.

Com esse método descrito, nós temos condições de avaliar o aumento do erro cometido pelos modelos criados devido ao fato de terem sido gerados com um conjunto de treinamento enviesado. Na próxima seção, descreveremos o método adotado para a implantação dos modelos criados de modo que eles possam realizar predições de uso dos dispositivos que compõem o smartphone de acordo com o padrão de comportamento do usuário.

5.6 IMPLANTAÇÃO DE MODELOS DE APRENDIZAGEM DE MÁQUINA

Após descrevermos os procedimentos para selecionar os atributos dos dispositivos que compõem o smartphone, construir os modelos inteligentes e validar a importância desses atributos para a capacidade de generalização, nós precisamos definir um método para que os modelos criados sejam implantados e possam fazer predições a respeito do uso dos dispositivos do smartphone. A seguir descrevemos as tarefas a serem realizadas para implantar os modelos e permitir que eles façam predição.

Disponibilizar modelo: Após a criação dos modelos usando os procedimentos descritos na seções anteriores, nessa tarefa o arquivo de modelos gerado é associado ao identificador do usuário. Como comentado na Seção 4.1, o identificador a ser usado será o IMEI. Após a disponibilização do modelo são disparadas duas tarefas que executam em paralelo:

- Uma tarefa para realizar as predições de uso dos dispositivos que compõem o smartphone que executa sempre que demandada.

- Uma tarefa para avaliar se o uso do modelo melhorou o tempo de uso do smartphone por parte do usuário.

Realizar predições de uso: Essa tarefa executa continuamente sempre que demandada. Após ter um modelo implantado, a aplicação de monitoramento demanda uma predição do servidor sempre que detectar que o usuário mudou o estado do smartphone, mudando o aplicativo em uso ou alterando o funcionamento de um dos dispositivos que compõem o smartphone.Para possibilitar que o servidor responsável pela criação e implantação dos modelos pudessem se comunicar com o smartphone para enviar a resposta às solicitações de predição de uso, nós utilizamos Representational State Transfer (REST).

Com esse método descrito, nós temos clareza a respeito da infraestrutura necessária para implantarmos os modelos criados e comunicar o servidor responsável por criar os modelos e fazer as predições de uso com o smartphone do usuário. Na próxima seção, estabeleceremos o método para monitorar a capacidade preditiva dos modelos gerados.

5.7 MONITORAMENTO E APRENDIZAGEM CONTÍNUA

De acordo com HUYEN(2022), a implantação de um modelo não é o final do processo de construção de nenhum sistema baseado em modelos de aprendizagem de máquina. Segundo a autora, o desempenho dos modelos degrada com o tempo e nós devemos, com isso, monitorar continuamente o desempenho deles para identificar problemas e implantar correções para corrigir o problema identificado.

HUYEN(2022) identifica como um dos principais problemas para degradação da capacidade preditiva dos modelos de aprendizagem de máquina o *drift* de dados. Conforme discutido na Seção 2.7, o *drift* de dados acontece quando a distribuição de probabilidade dos dados de entrada muda ao longo do tempo. Sabendo disso, nós definimos um método para monitorar e corrigir os modelos criados pelo módulo de criação de modelos. A seguir descrevemos as tarefas realizadas.

Avaliar modelo em uso: Durante o uso do modelo, o módulo responsável pelo monitoramento solicita ao aplicativo responsável pelo monitoramento do usuário, ao final do dia, o estado atual da bateria. Baseado nas atividades repassadas ao longo do dia e das predições feitas, o módulo de monitoramento dos modelos faz uma estimativa do uso da bateria e executa uma das seguintes ações:

- Caso a estimativa de carga restante da bateria feita pelo módulo de monitoramento ultrapasse o limiar de 25% de diferença, o módulo de monitoramento sinaliza a necessidade de treinamento de um novo modelo. Utilizamos o limiar de 25% porque boa parte das baterias atuais têm uma capacidade total de ao menos 3600mAh. Dessa forma, 25% de diferença em uma bateria com essa capacidade significa um total de 900mAh.
- Caso contrário, o modelo que está em uso continua sendo válido para o módulo de monitoramento.

Avaliar tempo de uso do modelo: Essa tarefa é executada em paralelo com a anterior para avaliar se o tempo que o modelo está sendo usado para estimar o uso de dispositivos que compõem o smartphone é superior a 30 dias. Baseado nessa análise, o módulo de monitoramento dos modelos realiza uma das seguintes ações:

- Caso o modelo esteja sendo usado em um período superior a 30 dias, o módulo de monitoramento de modelos notifica o aplicativo de monitoramento de uso para que seja coletado um novo uso do usuário sem a necessidade de criar um novo modelo. Esses novos dados coletados são avaliados pela tarefa responsável por avaliar o *drift* de dados.
- Caso contrário, o modelo de predição de uso continua a ser válido.

Avaliar drift de dados: Caso a tarefa responsável por avaliar o tempo de uso do modelo verifique que, embora com boa capacidade preditiva, o modelo já está em uso há um tempo superior a 30 dias, essa tarefa tem por finalidade avaliar os dados recém coletados em comparação com os dados históricos utilizados para gerar o modelo em uso. Essa comparação será feita utilizando ferramentas para avaliar o *drift* de dados existentes entre esses dois conjuntos: dados históricos e dados recém coletados. Caso seja verificado um *drift* superior a 20%, o módulo responsável pelo monitoramento dos modelos em uso sinaliza para que ocorra a criação de um novo modelo de forma preventiva.

5.8 CONSIDERAÇÕES FINAIS

Após realizar os estudos a respeito das estratégias para a medição do consumo energético do smartphone, nós podemos projetar um aplicativo Android que seja capaz de mensurar o consumo energético do smartphone de uma forma não tão intrusiva.

Os estudos a respeito dos dispositivos a serem monitorados nos permitiu entender melhor a relação do uso dos dispositivos presentes no smartphone e como esse uso se relaciona ao padrão de comportamento do usuário.

Estudar a respeito das estratégias adotadas na área de Engenharia de Machine Learning para coletar os dados, entendendo a respeito dos aspectos que devem ser observados como o escalonamento dos dados e o ajuste da distribuição desses dados, nos fez tornar a nossa abordagem mais robusta avaliando cada um dos atributos.

Após projetar esses módulos e investigar as ferramentas que devemos usar para realizar as atividades relacionadas ao desenvolvimento dos módulos para a criação dos modelos de uso, nós temos os procedimentos necessários para criar os modelos de uso dos dispositivos que compõem o smartphone baseado no padrão de comportamento do usuário.

No próximo capítulo, discutiremos a implementação dos módulos de software propostos nesta pesquisa.

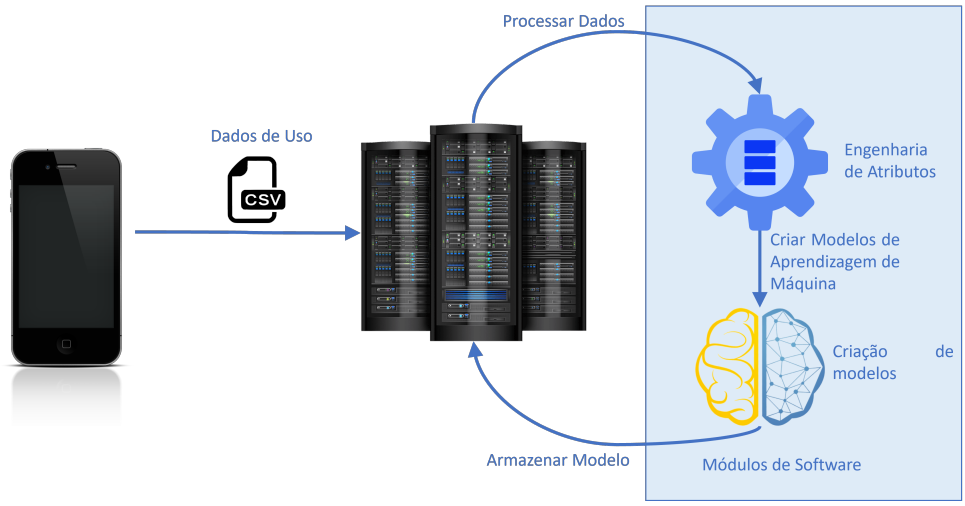
6 IMPLEMENTAÇÃO DOS MÓDULOS DE SOFTWARE

Neste capítulo, nós iremos discutir as estratégias adotadas para implementar os módulos de software propostos nesta pesquisa seguindo os procedimentos descritos no Capítulo 5.

6.1 VISÃO GERAL DA SOLUÇÃO

Nesta seção, iremos descrever uma visão geral da solução adotada para processar os dados de uso do usuário e criar os modelos de aprendizagem de máquina responsáveis por relacionar o padrão de comportamento do usuário com o uso dos dispositivos que compõem o smartphone. Então, nós iremos explicar como o aplicativo Android desenvolvido, responsável pela coleta dos dados de uso do usuário, se integra com os módulos de software propostos responsáveis por: processar os dados, realizar a engenharia de atributos, criar os modelos e armazená-los para realizar estimativas futuras. A visão geral da solução é mostrada na Figura 6. A seguir, descreveremos cada um dos passos principais.

Figura 6 – Visão Geral da Solução



Fonte: o autor (2022)

1. Enviar dados de uso: Ao final do período de coleta, o serviço Android envia os dados de uso do usuário para o servidor com uma identificação do usuário.

2. Processar os dados: O servidor processa os dados recebidos fazendo a descompressão do arquivo e fornecendo os dados para o módulo de engenharia de atributos.

3. Engenharia de Atributos: O primeiro passo é realizar a engenharia dos atributos e

preparar os dados para os algoritmos de aprendizagem de máquina que irão ser construídos pelo módulo de criação de modelos.

4. Criação de modelos: Após realizar a engenharia de atributos,o módulo de criação de modelos gera os modelos de aprendizagem de máquina, otimizando e treinando os algoritmos de aprendizagem de máquina mais apropriados.

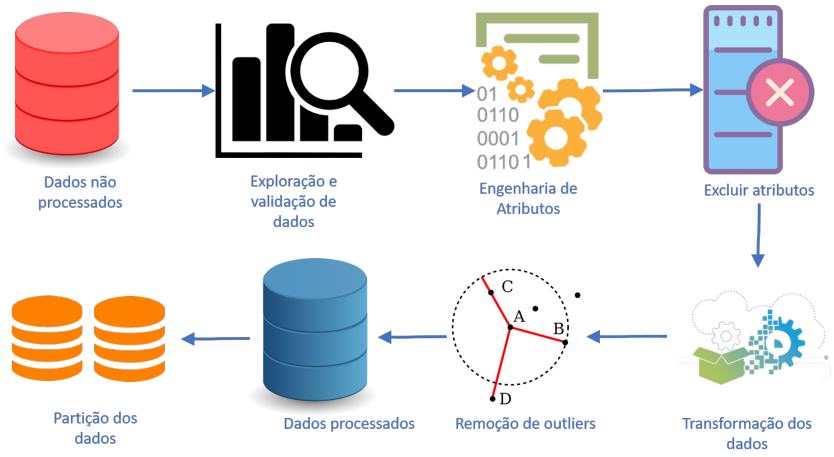
5. Armazenar modelo: Ao final, o modelo de aprendizagem de máquina mais apropriado para realizar futuras estimativas é armazenado.

Após apresentar a visão geral da solução, iremos discutir as estratégias adotadas para implementar os módulos discutidos na Seção 5.2.

6.2 MÓDULO DE ENGENHARIA DE DADOS

Como discutido na Seção 5.2, a atividade de preparação de dados é crucial para a construção de modelos de aprendizagem de máquina, pois ela é responsável por evitar a construção de modelos enviesados. Então, para implementar o módulo de engenharia de atributos proposto, nós adotamos o procedimento mostrado na Figura 7. A seguir, são descritos os passos adotados.

Figura 7 – Módulo de Engenharia de Dados



Fonte: o autor (2022)

1. Carregar os dados não processados: Nesse passo, o sistema carrega os dados de uso do usuário usando o método **load_csv** da biblioteca Pandas (COMUNITY, 2022).

2. Exploração e validação dos dados: Nesse passo, o sistema avalia os dados para encontrar valores nulos nas colunas usando os métdos **is_na()** e **any(axis=1)** do objeto DataFrame

do Pandas (COMUNITY, 2022).

3. Engenharia de Atributos: Com os dados previamente validados durante o passo de exploração, agora, o sistema realiza algumas transformações nos dados: gera a coluna ***power*** em mW multiplicando a corrente instantânea obtida em miliAmpère (mA) e a tensão em volts e calcula a intensidade de sinal móvel usando a relação $10^{intensidadeSinal/10}$.

4. Excluir Atributos: Usando a biblioteca *Feature-engine* (GALLI; SAMIULLAH; GALLI, 2022), nesse momento, o sistema exclui os atributos referentes à corrente instantânea(mA) e à tensão em Volts(V).

5. Tranformação dos dados: O módulo, nesse passo, filtra os atributos que possuem tipos de dados do tipo **int64** e **float64** e aqueles atributos que possuem outros tipos de dados diferentes desses para dividi-los em dois grupos (atributos categóricos e atributos numéricos), a fim de realizar a transformação dos dados. Além disso, nesse passo, o módulo transforma os atributos referentes à frequência de operação da CPU em atributos categóricos, uma vez que o ajuste de frequência é feito pelo Android de forma discreta. Usando o *OrdinalEncoder* (SCIKIT-LEARN, 2022f), o módulo aplica a transformação nos atributos categóricos e realiza a transformação dos atributos numéricos usando o *MinMaxScaler* (SCIKIT-LEARN, 2022e). Além disso, o módulo também aplica o *QuantileTransformer* (SCIKIT-LEARN, 2022g) para tornar as distribuições dos atributos numéricos mais próximas à distribuição gaussiana e, com isso, atender aos requisitos de alguns algoritmos de aprendizagem de máquina, como os baseados em modelos lineares.

6. Remoção de Outliers: Após aplicar as transformações necessárias, o passo seguinte do módulo é utilizar a classe *LocalOutlierFactor* (SCIKIT-LEARN, 2022c), da biblioteca Scikit-learn utilizando o esquema automático presente nessa classe que permite a ela escolher o melhor algoritmo a ser usado para remover os *outliers* entre as opções (ball_tree,kd_tree e busca por força bruta).

7. Dados Processados: Ao final dos procedimentos de carga de dados e remoção de *outliers*, o módulo retorna, neste momento, os dados processados a fim de serem divididos, conforme discutido na Seção 5.2.

8. Partição dos Dados: Nesse passo, com os dados processados, caso sejam usados para treinamento, o sistema particiona os dados em dois subconjuntos: conjunto de treinamento, conjunto de validação e conjunto de testes. Para implementar essa funcionalidade, nós consideramos o procedimento descrito na Seção 5.2.

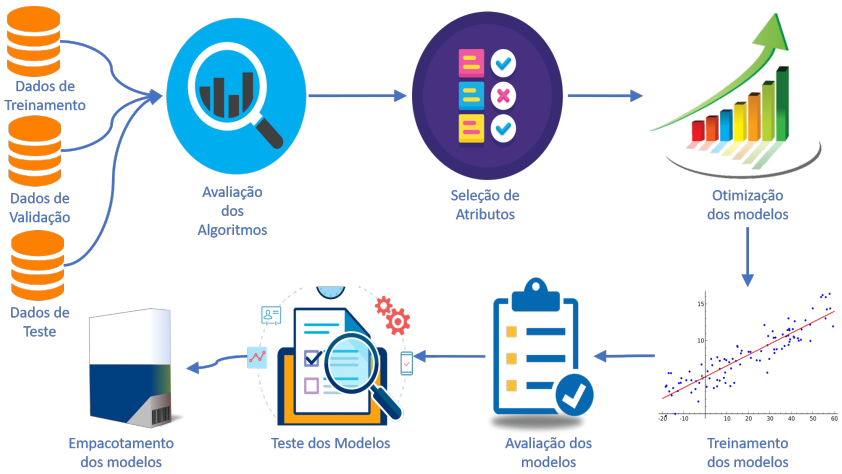
Com o módulo responsável pela engenharia de atributos implementado, nós podemos avaliar as estratégias mais apropriadas para implementar o módulo responsável por realizar a criação dos modelos, utilizando uma abordagem de testes de modelos, uma vez que os dados estão no formato apropriado para serem processados pelos algoritmos. A seção a seguir irá discutir as estratégias adotadas para implementar o módulo responsável por fazer a criação dos modelos.

6.3 MÓDULO DE CRIAÇÃO DE MODELOS

Como discutido na Seção 5.3, o processo de construção de um modelo de aprendizagem de máquina é um processo iterativo. Dessa forma, nós precisamos estabelecer um procedimento adequado para implementar esse módulo de software proposto.

Esse módulo será responsável por criar os modelos que irão auxiliar tanto os desenvolvedores de aplicativos Android quanto os mecanismos de otimização automática a serem usados pelos fabricantes. Tais mecanismos de otimização adotarão estratégias para escolher os dispositivos e as configurações mais adequadas, a fim de realizar uma tarefa de acordo com o padrão de comportamento do usuário. Os passos adotados para construir o módulo de criação de modelos são mostrados na Figura 8 e explicados a seguir.

Figura 8 – Módulo de Engenharia de Modelos



Fonte: o autor (2022)

1. Dados de Treinamento, Validação e Teste: Nosso módulo de software proposto obtém os dados de treinamento, validação e teste fornecidos pelo Módulo de Engenharia de Atributos para realizar a criação dos modelos de aprendizagem de máquina.

2. Avaliação dos Algoritmos: Conforme discutido na Seção 5.3, nessa etapa nós realizamos os procedimentos de **buscar regressores** e **selecionar regressores** avaliando vários algoritmos com os conjuntos de treinamento e teste providos para escolher aqueles mais proeminentes, usando um processo automatizado de avaliação dos algoritmos em suas configurações padrão. Para implementar esta funcionalidade em nosso módulo de criação de modelos proposto utilizando a estratégia automática de avaliação, nós nos baseamos na biblioteca LazyPredict (PANDALA; SILVA, 2022), fazendo nela os ajustes necessários para seguirmos os procedimentos decritos na Seção 5.3. Essa biblioteca analisa todos os regressores das bibliotecas Scikit-learn (PEDREGOSA et al., 2011), XGBoost (CHEN; GUESTRIN, 2016), e Lightgbm (KE et al., 2017). A partir dos resultados fornecidos pela biblioteca, nós escolhemos os 5 algoritmos mais bem classificados a serem usados com os dados apresentados.

3. Seleção de Atributos: Esse passo é opcional no *pipeline* do nosso módulo de criação de modelos aqui proposto. Como descrito na Seção 2.5, realizar seleção de atributos diminui o tempo para predição do modelo e – como discutido na Seção 2.3 – embora estejamos em um ambiente acadêmico, nos preocupamos em atender algumas recomentadações feitas pelo RAJ(2021) para o ambiente de produção. Nós implementamos esta funcionalidade dessa forma porque gostaríamos de avaliar o desempenho e o tempo para otimizar,treinar e realizar predições com os modelos gerados com e sem a seleção de atributos aplicados previamente nos conjuntos de dados de treinamento e testes. Para implementar a seleção de atributos em nosso módulo de criação de modelos de aprendizagem de máquina usando as três abordagens projetadas e explicadas perviamente, na Seção 5.3, nós utilizamos as classes ***RFE Feature Selection*** (SCIKIT-LEARN, 2022a) e ***SelectKBest***(SCIKIT-LEARN, 2022h) da biblioteca do Scikit-learn e o método ***GeneticSelectionCV*** da biblioteca sklearn-genetic(CALZOLARI, 2022).

4.Otimização dos modelos: Na Seção 5.1.4, discutimos o procedimento de avaliação de modelos. Um dos passos da avaliação de modelos é a otimização de hiperparâmetros usando o conjunto de validação. Para implementar a funcionalidade de otimização de hiperparâmetros dos modelos em nosso módulo de criação de modelos de aprendizagem de máquina, nós utilizamos a otimização Bayesiana, como discutido na Seção 2.4.2. Para usar a abordagem Bayesiana, nós adotamos o *framework* de otimização de hiperparâmetros Optuna (AKIBA et al., 2019). Nós adotamos esse *framework* de otimização de hiperparâmetros devido a duas características principais: a) pela possibilidade de salvar as tentativas de otimização em um banco de dados usando o conceito de experimento e b) pela flexibilidade de escolher a direção

de otimização. Durante a otimização, nós adotamos a estratégia proposta por NG *et al.* (1997) para previnir o sobreajuste no conjunto de validação, conforme descrito na Seção 5.2.

5. Treinamento dos modelos: Após otimizar os hiperparâmetros dos cinco modelos escolhidos no passo anterior, nós implementamos o treinamento de modelos. Para implementar esta funcionalidade, utilizamos a função ***train_test_split*** da biblioteca Scikit-learn aleatorizando as linhas do conjunto de dados antes de treinar os modelos. Nosso módulo de criação de modelos de aprendizagem de máquina, seguindo os procedimentos descritos na Seção 5.1.4, realiza trinta treinamentos de cada um dos algoritmos, sempre aleatorizando e particionando os dados antes de treinar cada um dos modelos.

6. Avaliação dos modelos: Implementamos a avaliação de modelos em cada uma das trinta iterações de treinamento de modelos. Então, ao final de cada iteração, cada modelo treinado é avaliado usando os conjuntos de treinamento e teste com a métrica MSE (SCIKIT-LEARN, 2022d). Ao final, o sistema provê o modelo com o menor erro para cada um dos cinco algoritmos previamente escolhidos.

7. Teste dos modelos: Com o melhor modelo para cada um dos cinco algoritmos, o módulo de criação de modelos de aprendizagem de máquina os avalia usando o conjunto de dados de teste e retorna o desempenho do modelo no conjunto de teste usando a métrica RMSE.

8. Empacotamento dos modelos: Desenvolvemos o módulo de criação de modelos de aprendizagem de máquina proposto utilizando a linguagem de programação Python. Então, escolhemos o formato Pickle (SPECS, 2022) para empacotar os modelos criados pelo nosso sistema e servi-los em um outro momento.

Os passos 4-7 descritos anteriormente foram implementados seguindo os procedimentos descritos na Seção 5.1.4, de modo a cumprir o objetivo do passo **Avaliar modelos** descrito na Seção 5.3 como parte do procedimento de criação de modelos. Em seguida o passo 8 é realizado para cumprir a etapa **Empacotar modelos** descrita na Seção 5.3.

Com o Módulo de Criação de Modelos implementado, nós podemos avaliar os modelos e o mecanismo automático de criação deles. Na seção a seguir, iremos discutir as estratégias adotadas para implantar e monitorar os modelos de aprendizagem de máquina gerados pelo nosso Módulo de Criação de Modelos.

6.4 MÓDULO DE IMPLANTAÇÃO E MONITORAMENTO DE MODELOS

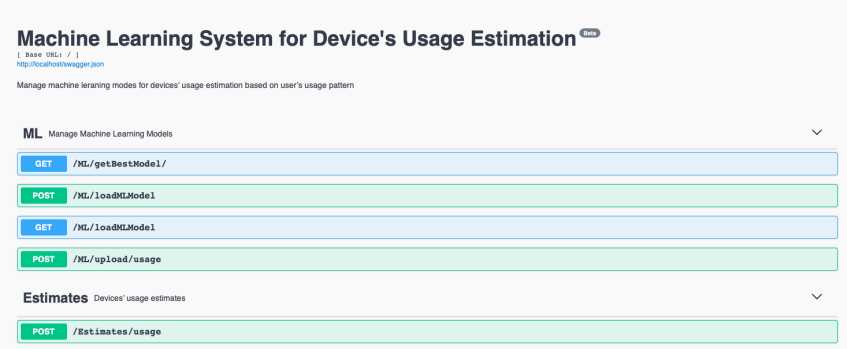
Com o serviço Android fornecendo os dados de utilização do usuário e os módulos de Engenharia de Atributos e Criação de Modelos tratando os dados e fornecendo os modelos adequadamente, nós precisamos definir um procedimento para comunicar os módulos de software desenvolvidos com o aplicativo Android.

Para atingir esse objetivo, nós implementamos o nosso módulo de implantação e monitoramento de modelos seguindo os procedimentos descritos nas Seções 5.6 e 5.7 conforme descrito a seguir.

1. Disponibilizar Modelos: Para disponibilizar os modelos, nós consideramos usar a arquitetura baseada em microsserviços por ser baseada em módulos de software leves que podem ser reiniciados rapidamente em caso de erro. Soluções baseadas em container são a estratégia mais adotada para a implantação de microsserviços e, dessa forma, nós adotamos esta abordagem para implantar os nossos microsserviços a fim de servir e monitorar os modelos inteligentes criados pelo nosso sistema usando a ferramenta docker-compose (DOCKER, 2022).

Para construir a API Rest, nós utilizamos um módulo responsável por fazer a interface entre o servidor WSGI Flask e o resto da aplicação. Escolhemos usar a biblioteca RestPlus (HAUSTANT, 2022) por conta da sua capacidade de prover uma interface *Swagger* para a documentação da API desenvolvida. Esta interface *Swagger* permite a um desenvolvedor usar a nossa API sem que ele precise ter acesso ao código fonte ou a qualquer documentação a respeito dela. A API desenvolvida é mostrada na Figura 9.

Figura 9 – Tela principal da API Rest desenvolvida



Fonte: o autor (2022)

A Figura 9 mostra a tela principal da API desenvolvida. Durante nossos experimentos, nós utilizamos as funções listadas para: realizar o upload do arquivo de registro de uso do usuário,

obter o melhor modelo para um cenário de uso e conseguir as estimativas de um modelo para um cenário de uso, cumprindo assim a etapa **Realizar predições de uso** descritas na Seção 5.6.

2. Avaliar modelo em uso: Conforme descrito na Seção 5.7, durante o monitoramento de modelos, temos que avaliar o modelo de uso dos dispositivos utilizando para isso duas estratégias:

- **Avaliar erro do modelo**: Utilizando o procedimento descrito na Seção 5.7, o módulo de implantação e monitoramento solicita ao aplicativo Android as informações necessárias para avaliar se a energia gasta corresponde às estimativas dadas pelo modelo em uso. Em caso de uma discrepância maior que 25%, o módulo de implantação e monitoramento solicita a coleta de novos dados de uso e criação de novos modelos de uso dos dispositivos que compõem o smartphone.

- **Avaliar tempo de uso**: Ao usar essa estratégia, verificamos a data de criação do arquivo do modelo que está em uso para averiguar se o arquivo foi gerado a mais de 30 dias a fim de solicitar ao aplicativo Android responsável pelo monitoramento de uso a realização de nova coleta para avaliarmos a existência de *drift* de dados.

3. Avaliar drift de dados: Gift e Deza (2021) destacam a importância de monitorar o *drift* de dados e a acurácia do modelo ao longo do tempo. Para monitorar o *drift* de dados, nós usamos a biblioteca Deepchecks (ROMANYSHYN, 2022).

Com a biblioteca Deepchecks nós podemos fazer diversas validações:

- Validar novos dados.

- Validar a partição de dados em treino e teste para avaliar se os dois subconjuntos resultantes são representativos.

- Realizar a validação do modelo após o treinamento.

Ao final da avaliação o módulo de implantação e monitoramento de modelos pode sinalizar ao Módulo de Criação de Modelos para criar um novo modelo utilizando a junção dos dados de uso do usuário utilizados previamente com os novos dados.

Após implementarmos as funcionalidades do Módulo de Implantação e Monitoramento de Modelos, finalizamos a construção de todos os módulos de software necessários para, juntamente com o aplicativo Android desenvolvido conforme descrito na Seção 7.2, criar modelos

que sejam capazes de estabelecer a relação entre o padrão de comportamento do usuário e o uso dos dispositivos que compõem o smartphone.

6.5 CONSIDERAÇÕES FINAIS

Após a implementação dos módulos de software propostos nesta pesquisa, temos os subsídios necessários para criar os modelos capazes de relacionar o padrão de comportamento do usuário à utilização dos dispositivos que compõem o smartphone.

Para avaliar o funcionamento e o desempenho do sistema proposto, utilizamos a infraestrutura de hardware e software descritas no próximo capítulo.

No próximo capítulo, descreveremos a infraestrtura de hardware e software utilizadas para avaliar o funcionamento do aplicativo Android e dos módulos de software desenvolvidos a fim de construir os modelos capazes de relacionar o padrão de comportamento do usuário ao uso dos dispositivos que compõem o smartphone.

7 INFRAESTRUTURA DE GERAÇÃO DE MODELOS

Após descrevermos, no Capítulo 6, como implementamos os módulos de software propostos nesta pesquisa, nós precisamos definir uma infraestrutura para a geração dos modelos de aprendizagem de máquina a partir do padrão de comportamento do usuário.

Neste capítulo iremos discutir a infraestrutura de monitoramento usada para coletar os dados do usuário. Além disso, discutiremos os experimentos realizados para: estabelecer os objetivos dos modelos que serão construídos para validar os módulos de software propostos, verificar a influência do número de amostras na capacidade preditiva dos modelos gerados, validar os requisitos necessários para a construção dos modelos, avaliar a capacidade preditiva dos modelos em conjunto de dados enviesados e avaliar a capacidade de nosso módulo de monitoramento de detectar o *drift* de dados a partir de um novo conjunto de dados de uso coletado.

7.1 INFRAESTRUTURA DE HARDWARE E SOFTWARE

Para definir uma infraestrutura de geração dos modelos de aprendizagem de máquina, precisamos criar uma infraestrutura de hardware e software que seja capaz de traçar um perfil dos dispositivos que compõem o smartphone descritos na Seção 5.1.2.

Dessa forma, a infraestrutura criada deve ser capaz de processar o uso dos dispositivos que compõem o smartphone de acordo com o padrão de comportamento do usuário e prover a informação necessária para criar modelos que relacionem o padrão de comportamento do usuário ao uso dos dispositivos que compõem o smartphone.

Para emular o comportamento de uso do usuário em diferentes configurações de uso, nós escolhemos o Motorola Moto G6(XT1925-3) e o Samusung Galaxy A10(A105M). Escolhemos esses smartphones por dois motivos principais:

1. Carcaterísticas do medidor de bateria: O Galaxy A10 possui um medidor de bateria acurado, enquanto o medidor de bateria do Moto G6 não atende a esse requisito, possibilitando a avaliação dos modelos gerados em conjuntos de dados enviesados.

2. Valor de mercado: Os dois smartphones possuem preços bastante acessíveis, na faixa de R$800 reais, sendo smartphones que são usados por grande parte da população.

De acordo com a Phone More(MORE, 2022), o Motorola Moto G6(XT1925-3) tem 32GB

de armazenamento interno e 3GB de memória RAM LPDDR3, usando o sistema operacional Android 8.0 (Oreo). Apesar da possibilidade de atualizar o sistema operacional, nós decidimos não alterá-lo para termos a possibilidade de testar a solução proposta nesta pesquisa em diferentes versões do Android.

O outro smartphone utilizado durante nossas emulações de uso foi o Samsung Galaxy A10, com 32GB de armazenamento interno, 2 GB de memória RAM e sistema operacional Android 9.0(Pie).

Para realizar o processamento dos dados de uso do usuário e a criação dos modelos inteligentes, nós usamos um computador Desktop com uma Graphical Processing Unit (GPU) dedicada Nvidia. O computador desktop utilizado possui as seguintes configurações: CPU Intel Core I3-8100, placa-mãe Asus TUF H310M-Plus Gamming, 24 GB de memória ram DDR4 2400MHZ e uma GPU Nvidia RTX 2060.

Para receber o arquivo de log de uso do usuário, nós usamos o *framework* de aplicação web Web Server Gateway Interface (WSGI) Flask (PROJECTS, 2022) com o *plugin* Flask RESTPlus (HAUSTANT, 2022). Essa infraestrutura composta pelo Flask e pelo Flask RESTPlus provê uma API para receber os dados de uso do usuário, processar esses dados, gerar os modelos de uso dos dispositivos e de consumo energético do smartphone e armazenar esses modelos no servidor com um identificador do usuário para ser utilizado em futuras estimativas.

Para analisar os dados recebidos, usamos a biblioteca de análise de dados Pandas (COMUNITY, 2022). Usamos o objeto DataFrame – que representa um conjunto de dados estruturado – e, com ele, convertemos as colunas e calculamos a potência em mW, usando a corrente e a tensão instantâneas registradas.

Após analisar os dados, temos os subsídios necessários para gerar os modelos que relacionam o padrão de comportamento do usuário ao uso dos dispositivos que compõem o smartphone usado por ele. Como discutido na Seção 5.3, nós optamos por utilizar os algoritmos de aprendizagem *shallow*.

Para construir os modelos de aprendizagem *shallow* e realizar as estratégias *wrapper* e de filtragem para selecionar atributos, utilizamos o *Scikit-learn*, discutido em Pedregosa *et al.* (2011), que é uma biblioteca de aprendizagem de máquina bem projetada para a linguagem de programação Python.

Após definir o hardware e o software necessários para realizar os experimentos, temos as condições necessárias para construir a nossa aplicação Android que será responsável por monitorar os dispositivos utilizados pelo usuário e o respectivo consumo energético do smartphone

devido a esse uso. Na próxima seção, discutiremos as abordagens adotadas para construir esse aplicativo de monitoramento e para avaliar a sobrecarga de consumo gerada por ele.

7.2 APLICATIVO DE MONITORAMENTO DO USO

Para implementar o aplicativo de monitoramento previamente projetado, conforme descrito no Apêndice B, usando as APIs relatadas, nós criamos um aplicativo Android denominado *Devices Monitor Service*.

O *Devices Monitor Service* possui monitores para os seguintes dispositivos: Bluetooth, CPU, Radio (3G/4G), Tela e WiFi. Além desses, nós criamos dois outros monitores: monitor de App e o monitor de Estado. O monitor de App é responsável por monitorar o aplicativo que está em uso pelo usuário.

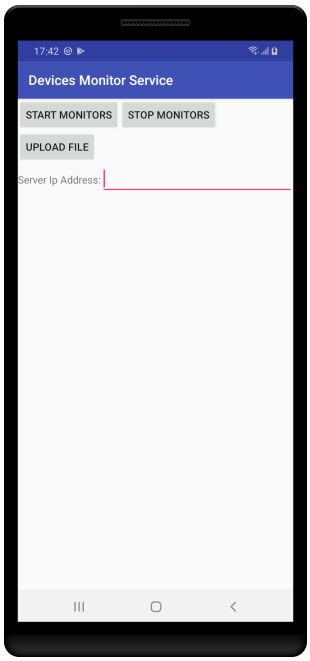
O monitor de Estado é responsável por sumarizar a informação de todos os outros monitores, por mensurar a corrente em mA e por medir a tensão em volts. Esse monitor é responsável também por coletar informações a respeito de todos os outros dispositivos, listados na Seção 5.1.2, que não possuem um monitor específico implementado.

Para coletar a informação a respeito dos outros dispositivos que não possuem um monitor implementado dentro do nosso aplicativo Android, o monitor de Estado interage com as ferramentas Linux descritas no Apêndice B através de um biblioteca de linha do comando chamada Android Shell (INC, 2021).

Os dados coletados pelo monitor de Estado são salvos em um arquivo Comma Separated Values (CSV) para transmiti-lo posteriormente. No momento da transmissão, o arquivo de log é comprimido usando a biblioteca MZip-Android(HANDYORG, 2022). Após comprimir o arquivo de log , o nosso aplicativo envia os dados ao nosso servidor, que os recebe e os descomprime para analisá-los em seguida. O *layout* do aplicativo desenvolvido é mostrado na Figura 10.

Para construir os modelos de consumo de forma acurada, precisamos mensurar a sobrecarga energética causada pelo nosso aplicativo de monitoramento, *Devices Monitor Service*. Para estimar o aumento de consumo devido ao nosso aplicativo, nós utilizamos o script *bash*, mostrado no Código 1.

Figura 10 – Aplicativo Devices Monitor Service



Fonte: o autor (2022)

Código Fonte 1 – Caracterizador de Consumo Energético

```
!#/bin/bash

count=0
while [ $count -lt 120 ]
do
current=`cat /sys/class/power_supply/battery/current_now_ua`
voltage=`cat /sys/class/power_supply/battery/voltage_now`
echo "$current;$voltage" >> log.txt
sleep 1
count=$((count+1))
done
```

Fonte: Elaborado pelo autor (2022)

Para mensurar a sobrecarga gerada pelo nosso aplicativo *Devices Monitor Service*, nós configuramos o Samsung Galaxy A10 em modo avião e realizamos 30 experimentos, cada um com 120 amostras em cada estado: sem o nosso aplicativo e com o nosso aplicativo. Então, para calcular o Intervalo de Confiança (IC) das médias da potência instantânea em cada uma das situações: com o aplicativo *Devices Monitor Service* executando e sem ele estar em execução, nós utilizamos o conceito de *Bootstrapping*. Construímos o intervalo de confiança usando as 30 médias obtidas a partir dos dados coletados anteriormente nos experimentos, com 1000 replicações para cada um dos cenários avaliados. Os resultados obtidos são mostrados

na Tabela 3.

Tabela 3 – Sobrecarga de consumo do aplicativo *Devices Monitor Service*

Estado do Aplicativo	Limite Inferior do IC	Limite Superior do IC
Em Execução	561.452mW	665.759mW
Fechado	402.635mW	447.131mW

Fonte: o Autor(2020)

A partir dos resultados mostrados na Tabela 3, percebemos que há uma diferença no consumo energético do smartphone quando o aplicativo está em execução. Podemos estimar,analisando os limites superiores dos intervalo de confiança, que a sobrecarga energética é da ordem de 200mW de potência instantânea.

Após desenvolver o aplicativo Android responsável por coletar os dados de uso do usuário e o respectivo consumo energético do smartphone, temos os subsídios necessários para avaliar os objetivos dos modelos que serão criados para avaliar o funcionamento dos módulos de software propostos nesta pesquisa. Na próxima seção discutiremos os modelos que decidimos criar para avaliar os módulos de software propostos nesta pesquisa e os critérios adotados para escolher quais atributos dos dispositivos que compõem o smartphone seriam selecionados para gerar os modelos.

7.3 OBJETIVOS DOS MODELOS

Após estabelecer o ambiente necessário para coletar dados de uso e gerar os modelos que relacionam o padrão de comportamento ao uso dos dispositivos que compõem o smartphone, nós precisamos estabelecer quais modelos criaríamos para avaliar o funcionamento dos módulos de software criados durante essa pesquisa.

Como mostrado no projeto do Módulo 2, responsável pela criação dos modelos de aprendizagem de máquina, nós projetamos um módulo de software capaz de gerar modelos para estimar o uso de qualquer dispositivo que compõe o smartphone baseado no padrão de comportamento do usuário. Na Seção 1.1, justificamos a existência desta pesquisa a partir de uma outra pesquisa anteriormante realizada a fim de entender a relação entre a frequência de operação da CPU e o consumo energético do smartphone. Dessa forma, para avaliar o funcionamento dos módulos de software propostos nesta pesquisa, nós decidimos criar modelos para estimar: a potência intantânea, uso da CPU e temperatura da CPU. A seguir nós justificamos

o motivo de escolha para estimar o uso de cada um dos dipositivos e da escolha dos atributos elencados para serem preditores dos modelos gerados.

1. Potência instantânea: Escolhemos criar um modelo para estimar a potência instantânea a partir do padrão de comportamento do usuário por sabermos, como discutido na Seção 5.1.1, que a potência instantânea se relaciona ao consumo energético do smartphone, motivação inicial desta pesquisa. Dessa forma, escolhemos como variável resposta do modelo o atributo **Power** em mW, calculado usando a correntre instantânea em mA e a tensão em Volts. Estabelecemos como atributos preditores para esse modelo todos os outros atributos selecionados dos dispositivos que compõem smartphone discutidos na Seção 5.1.2 e realizamos experimentos a fim de validar a importância desses atributos para a capacidade preditiva do modelo de predição de potência, como discutido na Seção 7.5.

Uso da CPU: Como discutido na Seção 1.1,a motivação inicial para esta pesquisa surgiu a partir de uma outra pesquisa que buscava relacionar a frequência de operação da CPU ao consumo energético do smartphone. De acordo com BRODOWSKI(2022), os *governors* intitulados *OnDemand* e *Interactive* operam ajustando a frequência da CPU de acordo com o uso dela. Os *governors* são políticas usadas pelo *kernel* do Linux, sistema no qual o Android se baseia, para atribuir a frequência de operação da CPU(BRODOWSKI, 2022).

Baseados nesse fato, ao pensarmos sobre os atributos que seriam necessários para gerar um modelo que estimasse o uso da CPU do smartphone baseado no padrão de comportamento, elencamos, como atributos preditores desse modelo, os seguintes atributos dos dispositivos que compõem o smartphone:Frequência da CPU, Estado da Tela, Estado do WiFi, Estado da rede móvel, Estado do Bluetooth, App em Uso, Período do dia e Orientação. A frequência de operação da CPU foi escolhida pela relação existente entre frequência e uso da CPU usada por alguns *governors*.

O estado da Tela foi escolhido porque diz muito a respeito do tipo de atividade que está sendo realizada pelo usuário, uma vez que ao desligar a tela, boa parte das atividades que estão sendo executadas não demandam tanto uso da CPU por estarem sendo executadas em segundo plano.

Já o estado das redes móvel e WiFi e o estado do Bluetooth foram escolhidos por também indicarem o tipo de atividade sendo executada pelo usuário já que em ambientes internos há uma maior probabilidade de uso das redes WiFi e do Bluetooth e para ambientes externos as redes móveis são usadas com maior frequência.

Os atributos App em Uso e Orientação foram escolhidos por possuírem, também, relação com a atividade que está sendo realizada pelo usuário. O aplicativo é o meio pelo qual o usuário faz uso de forma mais intensa da CPU e a orientação nos indica, por exemplo, se há um consumo de mídia ou jogos, atividades estas que costumam consumir mais CPU.

Já o Período do dia é um atributo que se relaciona à rotina do usuário, referindo-se a um uso maior ou menor do smartphone por parte do usuário e consequentemente a um uso maior da CPU.

Temperatura da CPU: Sabemos através da Física que a temperatura é uma das fontes de dissipação de potência e, consequentemente, de energia. Dessa forma, escolhemos gerar um modelo para estimar a temperatura da CPU. Para gerar o modelo que relaciona o padrão de comportamento do usuário à temperatura de operação da CPU, utilizamos todos os outros atributos selecionados dos dispositivos que compõem o smartphone, conforme discutido na Seção 5.1.2. Para avaliar, dentre os atributos selecionados, aqueles que mais conntribuíam para a capacidade preditiva do modelo, nós realizamos um experimento para validar os requisitos do modelo, conforme descrito na Seção 7.5.

Após estabelecermos os objetivos dos modelos a serem usados para avaliar os módulos de software propostos nesta pesquisa, nós precisávamos validar a importância dos atributos escolhidos como preditores para a construção dos modelos responsáveis por estimar a potência instantânea do smartphone e temperatura da CPU. No entanto, antes de procedermos para realizar essa validação dos requisitos, precisávamos avaliar quantas amostras seriam necessárias para gerar os modelos responsáveis por relacionar o comportamento do usuário com as variáveis resposta discutidas nessa seção. Na próxima seção, discutimos os procedimentos adotados para atingir esse objetivo.

7.4 AVALIAR O TAMANHO DA AMOSTRA

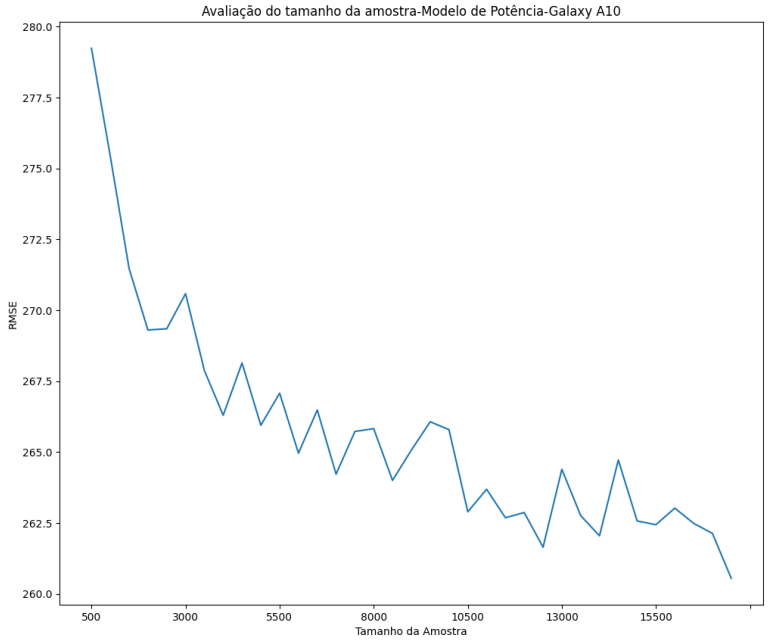
Na Seção 5.1.3, dicutimos a respeito de estudos que mostravam o quanto algoritmos distintos tiveram desempenho semelhante quando alimentados por uma quantidade de dados suficiente. Na Seção 5.1.3 estabelecemos um procedimento para verificar o número de amostras adequado para o nosso problema – criar modelos capazes de estabelecer a relação entre o padrão de comportamento do usuário e o uso dos dispostivos que compõem o smartphone.

Dessa forma, antes de procedermos com os experimentos para avaliar os módulos de

software criados nesta pesquisa, precisávamos estabelecer uma carga de trabalho capaz de emular o padrão de comportamento do usuário para gerarmos vários modelos alimentados com quantidades de dados distintas e verificarmos a influência do tamanho da amostra para a capacidade preditiva dos modelos gerados, seguindo o procedimento descrito anteriormente.

ELLIOTT; KOR; OMOTOSHO(2017) realizaram um trabalho em que mostram o consumo energético de diversos aplicativos comumente utilizados pelos usuários: WhatsApp, Facebook, Youtube, VLC, entre outros. Baseados nesse estudo, realizamos diversos experimentos utilizando o aplicativo Youtube, um dos aplicativos que mais consomem energia segundo os autores, com diferentes quantidades de amostras para avaliar a redução no erro de predição do modelo responsável por estimar a potência instantânea baseada no padrão de comportamento do usuário. Para realizar esse experimento, utilizamos todos os atributos preditores elencados na Seção 7.3. Como regressor base, conforme sugerido por BURKOV(2020), nós utilizamos o **KNN** com o número de vizinhos igual a 5 e realizamos a engenharia de atributos, conforme descrito na Seção 5.2. Os resultados são mostrados na Figura 11 e na Tabela 4.

Figura 11 – Relação entre o tamanho da amostra e erro de predição dos modelos para estimar potência instantânea



Fonte: o autor (2022)

A partir da Figura 11, podemos perceber que o erro de perdição do modelo diminui consideravelmente à medida que aumentamos o número de amostras. Outro fato que pudemos

perceber são as oscilações no erro de predição do modelo devido a dois fatores:

- estarmos construíndo os conjuntos de treinamento e testes com dados anteriormente aleatorizados,
- o processo de treinamento do modelo possui uma aleatoridade intrínseca.

Para nos apoiar em nossa conclusão a respeito do impacto gerado pelo tamanho da amostra no erro de predição do modelo, nós construímos a Tabela 4 para avaliarmos a diminuição do erro a cada 1 hora de experimento desde o tamanho inicial de 500 amostras.

Tabela 4 – Influência do tamanho da amostra no erro dos modelos para estimar potência instantânea

Tamanho da amostra	RMSE(mW) do melhor modelo
500	279.24
3500	267.88
7000	264.22
10500	262.90
14000	262.05
17500	260.55

Fonte: o Autor(2020)

A partir da análise da Tabela 4, podemos observar que há uma diminuição consistente a cada hora de experimento. Os resultados mostrados na Tabela 4 nos indicam a necessidade de termos experimentos de ao menos 5 horas ou mais, a fim de buscarmos construir modelos que sejam capazes de estabelecer melhor a relação entre o padrão de comportamento e potência instantânea, que em nosso estudo se relaciona ao uso da bateria.

A partir dessas constatações, realizamos experimentos emulando o uso do Youtube e do Chrome. Escolhemos o Youtube por ele apresentar um alto consumo de energia, conforme mostrado por ELLIOTT; KOR; OMOTOSHO(2017), e o Chrome por representar um carga semelhante ao WhatsApp em termos de transferência de dados e poder ter seu uso automatizado.

Na Seção 6.2 discutimos a respeito da manipulação de dados, no entanto, adotamos o procedimento de apenas manipular os dados quando o usuário tiver ao menos dois conjuntos de dados de uso, conforme discutido na Seção 4.2. Com isso, precisamos ter ao menos dois conjuntos de dados coletados para que o módulo de Engenharia de Atributos possa manipular os dados recebidos. Dessa forma, estabelecemos as duas cargas de trabalho necessárias para gerar os dois conjuntos de dados a serem usados inicialmente, conforme a Tabela 5.

Tabela 5 – Cargas de trabalho-Youtube e Chrome

Conjunto de Dados	Smartphone	Aplicativo	Tempo(s)
I	Galaxy A10	Chrome	634
		Youtube	11929
	Moto G6	Chrome	8467
		Youtube	30884
II	Galaxy A10	Youtube	11625
	Moto G6	Youtube	9350

Fonte: o Autor(2022)

Ao receber as duas cargas de trabalho, o módulo responsável por realizar a engenharia de atributos faz a junção dos arquivos, gerando um único conjunto de dados a ser usado para realizar a avaliação dos módulos de engenharia de atributos, criação de modelos e monitoramento.

Após realizarmos a análise do tamanho da amostra de dados necessária para gerarmos nossos modelos em nossa pesquisa, temos os subsídios necessários para realizar a outra tarefa citada na Seção 7.3: a validação dos requisitos dos modelos responsáveis por estimar a potência instantânea e a temperatura da CPU. Na próxima seção, discutiremos a respeito dos procedimentos adotados para atingir esse objetivo.

7.5 VALIDAR REQUISITOS

Na Seção 7.3 definimos os objetivos dos modelos a serem construídos para avaliar os módulos de software propostos nesta pesquisa. Após essa definição, a fim de validar o uso dos preditores elencados para construir os modelos responsáveis por estimar a potência instantânea do smartphone e a temperatura da CPU, verificamos a contribuição deles na capacidade preditiva dos modelos gerados. Antes de validarmos o uso dos preditores, precisávamos avaliar o tamanho da amostra necessária para construirmos os nossos modelos e gerarmos um conjunto de dados, conforme descrito na Seção 7.4.

Após construirmos o conjunto de dados a ser utlizado com o intuito de validar os preditores elencados para construir os modelos responsáveis por estimar a potência instantânea do smartphone e a temperatura da CPU, realizamos o procedimento descrito na Seção 5.4 para cada um dos modelos e obtivemos os resultados nas Tabelas 6 e 7.

Tabela 6 – Modelos para validar os requisitos- Erro dos modelos de potência em mW-Galaxy A10

Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
Bagging	235.65	238.83	238.17	239.54
LGBM	239.41	243.18	242.53	243.86
Extra Trees	249.51	254.79	254.04	255.47
XGB	321.77	329.52	328.41	330.58
Gradient Boosting	448.49	454.53	453.48	455.40

Fonte: o Autor(2022)

Tabela 7 – Modelos para validar os requisitos- Erro dos modelos de temperatura da CPU em $^{\circ}C$-Galaxy A10

Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
LGBM	0.64	0.67	0.66	0.67
Bagging	0.64	0.67	0.66	0.67
Extra Trees	0.82	0.85	0.85	0.86
XGB	1.02	1.04	1.04	1.05
Gradient Boosting	2.98	3.01	3.00	3.02

Fonte: o Autor(2022)

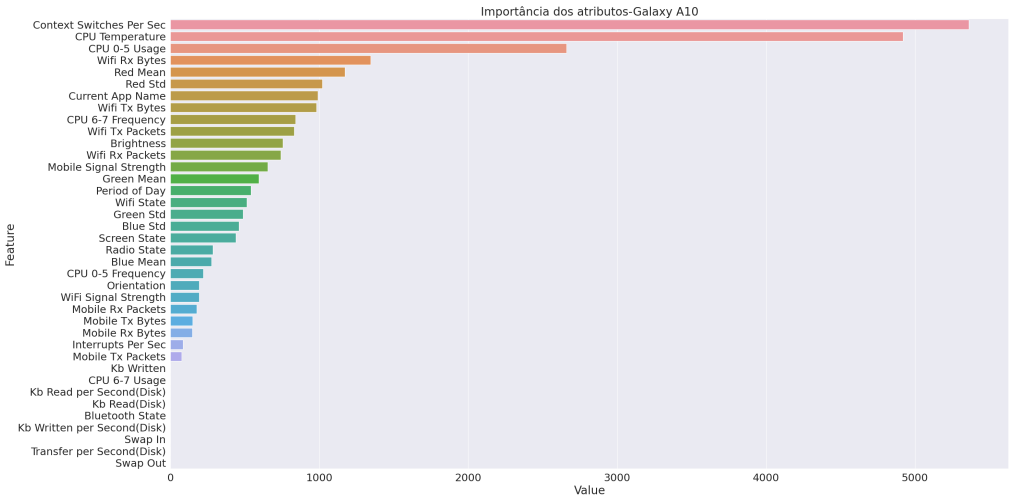
Os resultados mostrados nas Tabela 6 mostram que conseguimos construir os modelos responsáveis por estimar a potência instantânea com um erro apenas 35 mW acima do aumento de 200 mW obtido quando se usa o aplicativo para realizar o monitoramento de uso, conforme mostrado na Tabela 3 e comentado na Seção 7.2.

Já os resultados mostrados na Tabela 7 mostram que conseguimos construir modelos capazes de estimar a temperatura da CPU com um erro inferior a $1^{\circ}C$.

Baseados nos resultados obtidos, obtivemos a importância dos atributos preditores para a capacidade preditiva dos modelos responsáveis por estimar a potência instantânea do smartphone e a temperatura da CPU a fim de observar se, de fato, os atributos elencados como preditores contribuíram para a melhoria da capacidade preditiva dos modelos gerados. Os resultados obtidos são mostrados nas Figuras 12 e 13.

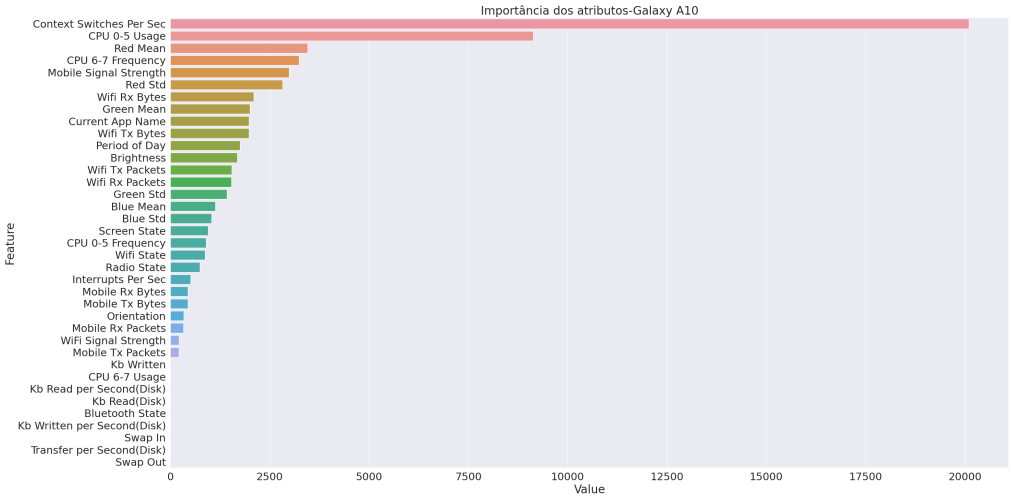
Os resultados mostrados nas Figuras 12 e 13 mostram que, de fato, boa parte dos atributos escolhidos contribuem para a capacidade preditiva dos modelos construídos, uma vez que possuem importância maior que 0. Nas figuras também é possível perceber que uma parte menor dos atributos elencados como preditores desses modelos não estão contribuindo para

Figura 12 – Validar Requisitos para os modelos de potência-Galaxy A10



Fonte: o autor (2022)

Figura 13 – Validar Requisitos para os modelos de temperatura da CPU-Galaxy A10



Fonte: o autor (2022)

a melhoria da capacidade preditiva deles, apenas aumentando a complexidade, porém, ao se fazer uso das técnicas de seleção de atributos, conseguimos eliminar esses atributos. Dessa forma, optamos por não eliminá-los a priori.

Após a construção dos modelos capazes de estimar a potência instantânea do smartphone e a temperatura da CPU, podemos verificar pelos resultados obtidos que os nossos módulos de engenharia de atributos e construção de modelos estão funcionando de forma adequada. Chegamos a esta conclusão a partir da análise dos resultados mostrados nas Tabelas 6 e 7. A partir da análise das Figuras 12 e 13, podemos avaliar que os atributos elecandos como preditores para os modelos responsáveis por estimar a potência instantânea do smartphone e

a temperatura da CPU de fato contribuem para a capacidade preditiva desses modelos.

Dessa forma, atingimos o outro objetivo – validar os atributos preditores para os modelos responsáveis por estimar a potência instantânea do smartphone e a temperatura da CPU – que estabelecemos após definir os requisitos dos modelos a serem construídos para validar os módulos de software propostos nesta pesquisa.

Após definirmos os objetivos dos modelos a serem construídos, precisávamos avaliar o erro cometido pelos modelos gerados em conjuntos de dados enviesados, a fim de avaliarmos o procedimento estabelecido na Seção 5.5 e a capacidade de nosso módulo de software de construção de modelos. Na próxima seção, descrevemos o experimento realizado para alcançar esse objetivo.

7.6 AVALIAÇÃO DOS MODELOS COM CONJUNTO DE DADOS ENVIESADOS

Como discutido na Seção 5.5, uma das maiores preocupações existentes quando se busca construir modelos de aprendizagem de máquina é a qualidade dos dados que serão usados. Alguns smartphones possuem medidores de bateria inacurados que não são capazes de atualizar a corrente instantânea mensurada por eles em intervalos regulares, gerando, assim, uma grande limitação dos trabalhos que buscam construir modelos de aprendizagem de máquina que estimam a potência instantânea do smartphone.

Diante dessa constantação, nós estabelecemos um método para construir modelos em conjuntos de dados enviesados, conforme descrito na Seção 5.5, e precisamos avaliar a qualidade dos modelos gerados com esse procedimento. Utilizando o nosso módulo de software responsável por criar modelos, conseguimos gerar modelos enviesados para estimar a potência instantânea seguindo o procedimento descrito anteriormente e obtivemosos resultados mostrados na Tabela 8.

A partir dos resultados mostrados na Tabela 8, podemos perceber que conseguimos obter modelos capazes de estimar a potência instantânea do smartphone, mesmo usando conjuntos de treinamento enviesados, com um erro 31 mW superior ao erro obtido quando não utilizamos um conjunto enviesado, como o mostrado na Tabela 6. A fim de avaliarmos o erro cometido por um regressor de base que apenas repete o último valor, como discutido na Seção 5.5, nós realizamos esse experimento e avaliamos o erro, obtendo um erro igual a 327.42 mW, sendo, portanto, superior ao erro obtido por quatro dos cinco modelos gerados pelo módulo de criação de modelos proposto nesta pesquisa.

Tabela 8 – Avaliação dos modelos em conjuntos de dados enviesados

Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
LGBM	266.93	266.93	266.93	266.93
HistGradient Boosting	268.70	269.57	269.37	269.78
Lasso Lars IC	277.60	277.60	277.60	277.60
OMP	280.40	280.40	280.40	280.40
Gradient Boosting	429.42	429.57	429.53	429.60

Fonte: o Autor(2022)

Após avaliarmos o módulo de criação de modelos quanto a capacidade de criar modelos que sejam capazes de realizar estimativas mesmo quando treinados em conjunto de dados enviesados, nós finalizamos a avaliação dos módulos de engenharia de atributos e criação de modelos.

Outro módulo criado por nós durante esta pesquisa é o módulo de monitoramento que tem por finalidade monitorar o uso dos modelos gerados e avaliar a necessidade de coletar novos dados e retreinar os modelos previamente gerados. Na próxima seção, discutiremos o experimento realizado para avaliar o módulo de monitoramento criado nesta pesquisa.

7.7 AVALIAÇÃO DO *DRIFT* DE DADOS

Como discutido na Seção 5.7, um dos principais fatores que degradam a capacidade preditiva dos modelos de aprendizagem de máquina é o *drift* de dados. Dessa forma, construímos um módulo de monitoramento dos modelos a fim de detectar a ocorrência do *drift* de dados utilizando um terceiro conjunto de dados.

YOON et al.(2012) realizou um estudo mostrando o consumo energético do aplicativo Google Maps. Baseado nesse estudo, decidimos realizar experimentos emulando uma viagem com o Google Maps enquanto escutávamos música usando o aplicativo Deezer Music. Nós avaliamos a execução do aplicativo Deezer Music em paralelo para avaliar também a capacidade de nossos modelos gerados capturarem a influência de uso de um aplicativo mesmo quando ele está em segundo plano. A carga de trabalho utilizada para gerar o conjuto de dados coletado é mostrada na Tabela 9.

Tabela 9 – Cargas de trabalho-Google Maps e Deezer Music

Conjunto de Dados	Smartphone	Aplicativo	Tempo(s)
III	Galaxy A10	Google Maps	11234
		Deezer Music	11234
	Moto G6	Google Maps	13500
		Deezer Music	13500

Fonte: o Autor(2022)

Após coletarmos os dados, utilizamos o nosso módulo de monitoramento para verificar a existência de *drift* de dados entre o conjunto unificado das cargas de trabalho mostradas na Tabela 5 e o conjunto de dados criado a partir da carga de trabalho mostrada na Tabela 9. Como descrito anteriormente, o módulo de monitoramento utiliza a biblioteca Deepchecks (ROMANYSHYN, 2022) para verificar a existência de *drift* de dados em dois conjuntos de dados existentes. Ao realizar a verificação filtrando os atributos para mostrar apenas aqueles que possuem *drift* maior que 0.2(20%), obtivemos os resultados mostrados na Tabela 10 para os smartphones Galaxy A10 e Moto G6.

Tabela 10 – Quantidade de atributos que apresentaram *drift* de dados

Smartphone	Número de atributos
Galaxy A10	38
Moto G6	34

Fonte: o Autor(2020)

A partir dos resultados mostrados na Tabela 10, o módulo de monitoramento proposto nesta pesquisa sinaliza a necessidade de retreinar o modelo utilizado a partir de um novo conjunto de dados formado pelas cargas de trabalho mostradas nas Tabelas 5 e 9.

Após avaliarmos o funcionamento do módulo de monitoramento dos modelos de aprendizagem de máquina quanto a capacidade de detectar o *drift* de dados e sinalizar a necessidade da criação de novos modelos, nós finalizamos a avaliação dos três módulos de software propostos nesta pesquisa para criar e monitorar modelos de aprendizagem de máquina que sejam capazes de relacionar o padrão de comportamento do usuário com o uso dos dispositivos que compõem o smartphone.

7.8 CONSIDERAÇÕES FINAIS

Após definirmos a infraestrutura de hardware e software necessária para implantar e avaliar os módulos de software propostos nesta pesquisa, nós podemos criar e monitorar modelos que sejam capazes de estabelecer a relação entre o padrão de comportamento do usuário e o uso dos dispositivos que compõem o smartphone.

Neste capítulo descrevemos o processo de criação do aplicativo Android que tem por finalidade coletar os dados de uso do usuário e avaliamos a sobrecarga energética causada pela execução do mesmo.

Também descrevemos, neste capítulo, o método adotado para estabelecer os objetivos dos modelos que serão criados a fim de avaliar os módulos de engenharia de atributos e criação de modelos com as mais diversas finalidades. Realizamos, também, uma avaliação a respeito da quantidade de amostras necessárias para que possamos criar modelos que tenham um menor erro de predição a fim de representar de forma mais fidedigna a relação existente entre o padrão de comportamento do usuário e a utilização dos dispositivos que compõem o smartphone.

Após definirmos os objetivos dos modelos que seriam gerados, nós validamos o uso dos atributos selecionados dos dispositivos monitorados como preditores desses modelos a fim de avaliar a importância deles para a capacidade preditiva dos modelos gerados. Em seguida, realizamos outra avaliação muito importatante: a avaliação dos modelos gerados em conjuntos de dados enviesados. Essa avaliação objetiva nos foi útil para verificarmos a qualidade dos modelos gerados pelo nosso módulo de criação de modelos.

Neste capítulo também descrevemos os experimentos realizados para avaliar o funcionamento do módulo de monitoramento proposto nesta pesquisa e a capacidade desse módulo detectar a existência de *drift* de dados e sinalizar o retreinamento dos modelos com um novo conjunto de dados. No próximo capítulo, descreveremos cenários de teste criados para avaliar os modelos gerados utilizando os conjuntos de dados construídos utilizando as cargas de trabalho mostradas nas Tabelas 5 e 9.

8 ESTUDOS DE CASO

Para avaliar os módulos de software responsáveis por realizar a engenharia de atributos, a criação, implantação e monitoramento de modelos, nós definimos uma abordagem composta de quatro cenários descritos na Tabela 11. Implementamos os módulos de engenharia de atributos e criação de modelos de forma flexível, permitindo-nos avaliar cada modelo que desejássemos.

Tabela 11 – Cenários Avaliados

Cenário	Descrição
I-Sem seleção de atributos	Neste cenário, consideramos todos os atributos preditores coletados pelo nosso serviço Android.
II-Com seleção de atributos usando Informação Mútua	Neste cenário, consideramos todos os atributos preditores coletados pelo nosso serviço Android e fizemos uso da técnica de filtragem usando informação mútua para selecionar os atributos mais relevantes.
III-Com seleção de atributos usando Algoritmos Genéticos	Neste cenário, consideramos todos os atributos preditores coletados pelo nosso serviço Android e fizemos uso de Algoritmos Genéticos para selecionar os atributos mais relevantes.
IV-Com seleção de atributos usando Eliminação Recursiva de Atributos	Neste cenário, consideramos todos os atributos preditores coletados pelo nosso serviço Android e fizemos uso da técnica de Eliminação Recursiva de Atributos para selecionar os atributos mais relevantes.

Fonte: o Autor(2022)

Nós avaliamos os modelos empacotados, conforme descrito na Seção 5.1.4. Após o empacotamento dos modelos para cada cenário avaliado, implementamos uma função que constrói uma tabela usando o objeto DataFrame da biblioteca Pandas (COMUNITY, 2022), com o desempenho de cada modelo no conjunto de dados de teste, classificando-os com base em seus respectivos desempenhos.

Os dados que utilizamos para avaliar o desempenho dos modelos gerados pelo módulo de criação de modelos proposto nesta pesquisa foram os dados das cargas de trabalho mostradas nas Tabelas 5 e 9. A seguir, avaliaremos os modelos gerados para estimar a potência instantânea nos cenários mostrados na Tabela 11, usando as cargas de trabalho mostradas na Tabela 5.

8.1 AVALIAÇÃO DOS MODELOS DE POTÊNCIA PARA YOUTUBE E CHROME

Para avaliarmos os modelos estimadores da potência instantânea gerados pelos nossos módulos de software propostos, seguimos o procedimento descrito na Seção 5.1.4 usando, primeiramente, as cargas de trabalho mostradas na Tabela 5. Os resultados obtidos usando o conjunto de testes para cada smartphone avaliado, Galaxy A10 e Moto G6, são mostrados nas Tabelas 14 e 15. Nessas tabelas, nós podemos: a) avaliar a média dos erros dos 30 modelos gerados para cada algoritmo avaliado usando a métrica RMSE e b) classificar os erros para escolher o modelo com o menor erro. Todos os erros são expressos em mW.

Para mensurar o quão bons esses resultados de erro dos estimadores gerados pelo nosso módulo são, precisamos analisar as estatísticas do conjunto de dados de testes para os smartphones Galaxy A10 e Moto G6. Os resultados são mostrados na Tabela 12 em mW.

Tabela 12 – Estatísticas do conjunto de dados de teste para potência instantânea(mW)

Smartphone	25° Percentil	Mediana	75° Percentil	IIQ
Galaxy A10	962.26	1214.78	1425.23	522.97
Moto G6	384.40	1286.60	1775.00	1390.60

Fonte: o Autor(2022)

Como discutido na Seção 2.3, embora estejamos em um ambiente acadêmico, nos preocupamos também em atender a algumas das necessidades presentes no ambiente de mercado. Com base nisso, durante a geração dos modelos, o nosso módulo de criação de modelos proposto calcula o tempo necessário para realizar todo o procedimento descrito na Seção 5.3. Os resultados obtidos são mostrados na Tabela 13

Tabela 13 – Tempos para construção dos modelos de potência por cenário

Cenário	Tempo(s)	
	Galaxy A10	Moto G6
I	5166	5426
II	3340	2719
III	1699	5702
IV	6110	2716

Fonte: o Autor(2022)

Como discutido na Seção 2.5, podemos perceber que dos três cenários avaliados em que fazemos seleção de atributos houve reduções significativas no tempo para otimizar os hiper-

parâmetros, treinar e testar os modelos escolhidos.

Tabela 14 – Modelos para estimar potência instantânea(mW)- Erro dos modelos-Galaxy A10

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**Bagging**	233.52	238.06	237.30	238.76
	LGBM	236.33	239.93	239.26	240.55
	HistGradient Boosting	243.45	246.54	245.97	247.14
	XGB	274.07	282.99	281.86	284.02
	Gradient Boosting	426.55	430.91	430.01	431.90
II	**LGBM**	234.76	237.35	236.87	237.80
	HistGradient Boosting	247.87	251.43	250.70	252.30
	Lasso Lars IC	253.25	257.55	256.95	258.10
	XGB	298.94	302.71	301.97	303.45
	Gradient Boosting	423.39	428.16	427.20	429.04
III	**LGBM**	235.86	239.42	238.93	239.89
	HistGradient Boosting	245.29	249.29	248.70	249.89
	Bayesian Ridge	254.48	257.34	256.73	257.92
	XGB	262.15	266.86	266.15	267.56
	Gradient Boosting	424.15	428.49	427.70	429.20
IV	**LGBM**	234.94	238.05	237.49	238.58
	XGB	248.19	253.31	252.63	253.99
	Ridge	254.08	258.21	257.63	258.76
	Linear Regression	254.72	257.58	256.95	258.21
	Gradient Boosting	424.33	429.46	428.55	430.43

Fonte: o Autor(2022)

Observando os resultados mostrados na Tabela 14, podemos observar duas características relevantes:

- Os modelos mais bem classificados em cada um dos cenários estudados não apresentam

diferença significativa, menos de 43.05% de diferença entre os erros em cada um dos cenários avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

- Com exceção do cenário **I**, no qual não aplicamos nenhuma estratégia de seleção de atributos, os melhores modelos elencados são obtidos a partir do algoritmo LGBM.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de, em todos os cenários avaliados, haver uma predominância dos modelos baseados em *ensembles* de árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 2.3.3, os algoritmos baseados em árvores terem grande poder de generalização devido ao fato de eles avaliarem, durante o processo de treinamento, os atributos que mais reduzem a variância da variável resposta para estabelecer os nós de divisão da árvore de regressão.

A partir dos resultados observados, como a diferença entre os melhores modelos em cada um dos cenários avaliados não foi considerada significativa, o módulo de geração de modelos escolhe o melhor modelo do cenário com o menor tempo, o cenário **III** na análse em questão.

Observando os resultados mostrados na Tabela 15, podemos verificar duas características relevantes:

- Os modelos mais bem classificados em cada um dos cenários estudados não apresentam diferença significativa, menos de 108.08% de diferença entre os erros em cada um dos cenários avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

- Em todos os cenários avaliados os melhores modelos elencados são obtidos a partir do algoritmo **Extra Tree**.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de, em todos os cenários avaliados, haver uma predominância dos modelos baseados em *ensembles* de árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 2.3.3, os algoritmos baseados em árvores terem grande poder de generalização devido ao fato de eles avaliarem, durante o processo de treinamento, os atributos que mais reduzem a variância da variável resposta para estabelecer os nós de divisão da árvore de regressão.

A partir dos resultados observados, como a diferença entre os melhores modelos em cada um dos cenários avaliados não foi considerada significativa, o módulo de geração de modelos escolhe o melhor modelo do cenário com o menor tempo, o cenário **IV** na análse em questão.

Conforme discutido na Seção 1.1, esta pesquisa se originou a partir da necessidade de modelar o consumo energético do smartphone a partir do padrão de comportamento do usuário.

Tabela 15 – Modelos para estimar potência instantânea(mW)- Erro dos modelos-Moto G6

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**Extra Tree**	113.66	125.61	123.25	127.74
	Decision Tree	114.56	125.35	123.56	127.27
	Bagging	292.38	297.08	296.47	297.73
	Extra Trees	373.78	378.11	377.56	378.68
II	**Extra Tree**	89.18	101.72	100.00	103.58
	Decision Tree	95.84	102.42	101.05	103.76
	Bagging	292.38	252.85	252.27	253.42
	Extra Trees	503.39	556.29	548.69	564.06
III	**Extra Tree**	94.86	101.95	100.35	103.44
	Decision Tree	94.93	101.87	100.54	103.22
	Bagging	243.41	249.89	249.16	250.59
	Extra Trees	381.39	384.65	383.83	385.49
IV	**Extra Tree**	94.73	101.11	99.99	102.15
	Decision Tree	96.14	103.45	101.64	105.11
	Extra Trees	398.47	408.82	407.57	410.25

Fonte: o Autor(2022)

Com isso, precisávamos avaliar as fontes de variabilidade nos dados existentes para contribuir para o erro dos modelos escolhidos para os smartphones Galaxy A10 e Moto G6.

A partir da análise da sobrecarga energética gerada pelo aplicativo, conforme mostrado na Tabela 3 e discutido na Seção 7.2, percebemos que houve uma sobrecarga da ordem de 200 mW e o modelo escolhido para realizar a estimativa da potência instantânea possui um erro de 235 mW para o Galaxy A10, cometendo, portanto, um erro de 35 mW a mais que a sobrecarga do aplicativo. A partir desses resultados, pudemos concluir que, para o nosso propósito, conseguimos atingir o nosso objetivo de estimar a potência instantânea a partir do padrão de comportamento do usuário. Na próxima seção, avaliaremos os modelos gerados pelo nosso módulo de criação de modelos proposto para estimar a temperatura e o uso da CPU.

8.2 AVALIAÇÃO DOS MODELOS DE USO E TEMPERATURA DA CPU PARA YOUTUBE E CHROME

Após a construção dos modelos para estimar a potência instantânea do smartphone, conforme discutido na Seção 7.3, nós precisamos construir modelos para estimar o uso e a temperatura da CPU, usando, primeiramente, as cargas de trabalho mostradas na Tabela 5, a fim de mostrar a flexibilidade de construção de modelos pelos módulos de softwares propostos nesta pesquisa.

Os resultados obtidos pelos modelos responsáveis por estimar a temperatura da CPU usando o conjunto de testes para cada smartphone avaliado, Galaxy A10 e Moto G6, são mostrados nas Tabelas 18 e 19. Nessas tabelas, nós podemos: a) avaliar a média dos erros dos 30 modelos gerados para cada algoritmo avaliado usando a métrica RMSE e b) classificar os erros para escolher o modelo com o menor erro. Todos os erros são expresos em $^{\circ}C$.

Para mensurar o quão bons esses resultados de erro dos estimadores gerados pelo nosso módulo são, precisamos analisar as estatísticas do conjunto de dados de testes para os smartphones Galaxy A10 e Moto G6. Os resultados são mostrados na Tabela 16 em $^{\circ}C$.

Tabela 16 – Estatísticas do conjunto de dados de teste para temperatura da CPU($^{\circ}C$)

Smartphone	25° Percentil	Mediana	75° Percentil	IIQ
Galaxy A10	38.00	39.00	41.00	3.00
Moto G6	31.00	34.50	36.20	5.20

Fonte: o Autor(2022)

Para estimar a temperatura da CPU, analisamos os mesmos cenários utilizados para avaliar os modelos estimadores da potência instantânea. Dessa forma, os tempos necessários para realizar todo o procedimento descrito na Seção 5.3 foram calculados e os resultados obtidos são mostrados na Tabela 17.

Como discutido na Seção 2.5, podemos perceber que, dos três cenários avaliados em que fazemos seleção de atributos, houve reduções significativas, novamente, no tempo para otimizar os hiperparâmetros, treinar e testar os modelos escolhidos.

Observando os resultados mostrados na Tabela 18, podemos observar três características relevantes:

- Em três dos quatro cenários avaliados, os modelos elencados com o menor erro possuem o mesmo erro, 0.64%.

Tabela 17 – Tempos para construção dos modelos de temperatura da CPU por cenário

Cenário	Tempo(s)	
	Galaxy A10	Moto G6
I	3438	13121
II	2936	4489
III	2996	8355
IV	1166	5528

Fonte: o Autor(2020)

- O modelo mais bem classificado no cenário **IV** não apresenta diferença significativa em relação aos outros modelos, menos de 7.69% de diferença entre os erros em cada um dos cenários avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

- Em todos os cenários avaliados, os melhores modelos elencados são obtidos a partir do algoritmo LGBM.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de, em todos os cenários avaliados, novamente, haver uma predominância dos modelos baseados em *ensembles* de árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

A partir dos resultados observados, como a diferença entre os melhores modelos em cada um dos cenários avaliados não foi considerada significativa, o módulo de geração de modelos escolhe o melhor modelo do cenário com o menor tempo, o cenário **IV** na anâlse em questão.

Observando os resultados mostrados na Tabela 19, podemos observar três características relevantes:

- Em três dos quatro cenários avaliados, os modelos elencados com o menor erro possuem o mesmo erro, 0.04%.

- O modelo mais bem classificado no cenário **I** apresenta diferença significativa em relação aos outros modelos, mais de 15.07% de diferença entre os erros em cada um dos cenários avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

- Em todos os cenários avaliados, os melhores modelos elencados são obtidos a partir do algoritmo **Decision Tree**.

Tabela 18 – Modelos para estimar Temperatura da CPU($^{\circ}C$)- Erro dos modelos-Galaxy A10

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**LGBM**	0.64	0.66	0.65	0.66
	Bagging	0.65	0.68	0.67	0.68
	Extra Trees	0.83	0.87	0.86	0.87
	HistGradient Boosting	0.87	0.90	0.89	0.90
	XGB	0.88	0.90	0.90	0.91
II	**Bagging**	0.64	0.66	0.66	0.67
	LGBM	0.67	0.69	0.68	0.69
	HistGradient Boosting	0.74	0.77	0.76	0.77
	Extra Trees	0.84	0.86	0.86	0.87
	XGB	0.96	0.98	0.97	0.98
III	**LGBM**	0.64	0.66	0.65	0.66
	Bagging	0.65	0.67	0.67	0.68
	Extra Trees	0.81	0.87	0.86	0.89
	XGB	1.04	1.06	1.05	1.06
	HistGradient Boosting	1.32	1.35	1.34	1.36
IV	**LGBM**	0.67	0.70	0.69	0.70
	Decision Tree	0.77	0.78	0.78	0.79
	Extra Trees	0.82	0.88	0.87	0.89
	XGB	0.96	0.98	0.98	0.98
	Gradient Boosting	2.76	2.79	2.78	2.79

Fonte: o Autor(2022)

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de, em todos os cenários avaliados, novamente, haver uma predominância dos modelos baseados em árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

A partir dos resultados observados, como a diferença entre os melhores modelos em três dos quatro cenários avaliados não foi considerada significativa, o módulo de geração de modelos escolhe o melhor modelo do cenário com o menor tempo, o cenário **II** na análse em questão.

Tabela 19 – Modelos para estimar Temperatura da CPU($^{\circ}C$)- Erro dos modelos-Moto G6

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**Decision Tree**	0.05	0.06	0.06	0.06
	Extra Tree	0.05	0.06	0.06	0.06
	Bagging	0.16	0.17	0.16	0.17
	Extra Trees	0.91	0.98	0.97	0.99
II	**Decision Tree**	0.04	0.04	0.04	0.05
	Extra Tree	0.04	0.05	0.05	0.05
	Bagging	0.12	0.13	0.12	0.13
	Extra Trees	1.04	1.28	1.24	1.33
III	**Decision Tree**	0.04	0.04	0.04	0.04
	Extra Tree	0.04	0.04	0.04	0.04
	Bagging	0.13	0.14	0.14	0.14
	Extra Trees	1.00	1.10	1.08	1.12
IV	**Decision Tree**	0.04	0.04	0.04	0.04
	Extra Tree	0.04	0.05	0.04	0.05
	Extra Trees	0.90	0.97	0.95	0.98

Fonte: o Autor(2022)

Com base nas análses do modelos estimadores da temperatura da CPU, podemos notar que o erro é menor que $1^{\circ}C$, resultando em um erro bastante baixo quando comparado à variabilidade dos dados, $3^{\circ}C$ para o Galaxy A10 e $4^{\circ}C$ para o Moto G6.

Em seguida realizamos a análise dos erros cometidos pelos modelos responsáveis por estimar o uso da CPU baseado no padrão de comportamento do usuário. Para construir os modelos estimadores do uso da CPU, nós não avaliamos o erro cometido em cenários com seleção de atributos pois selecionamos os atributos preditores, conforme discutido na Seção 7.3. Dessa forma, os erros cometidos pelos estimadores do uso da CPU são mostrados nas Tabelas 21 e 22. As estatísticas do conjunto de testes usado para avaliar o erro dos modelos são mostradas na Tabela 20.

Observando os resultados mostrados na Tabela 21, podemos observar que os modelos não apresentam diferença significativa entre si, menos de 52.17% de diferença entre os erros em

Tabela 20 – Estatísticas do conjunto de dados de teste para uso da CPU(%)

Smartphone	25° Percentil	Mediana	75° Percentil	IIQ
Galaxy A10	26.00	46.00	50.00	24.00
Moto G6	5.00	11.00	13.00	8.00

Fonte: o Autor(2022)

Tabela 21 – Modelos para estimar Uso da CPU(%)- Erro dos modelos-Galaxy A10

Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
Bagging	5.32	5.55	5.51	5.58
LGBM	5.34	5.53	5.49	5.56
Decision Tree	5.36	5.53	5.50	5.55
HistGradient Boosting	5.87	6.05	6.02	6.07
XGB	7.35	7.45	7.43	7.47

Fonte: o Autor(2020)

cada um dos modelos avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de todos os modelos avaliados, novamente, se basearem em árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

Tabela 22 – Modelos para estimar Uso da CPU(%)- Erro dos modelos-Moto G6

Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
LGBM	3.82	3.94	3.92	3.96
HistGradient Boosting	3.85	3.98	3.96	3.99
Decision Tree	3.89	3.95	3.94	3.96
Extra Trees	3.89	3.97	3.95	3.99
XGB	4.66	4.77	4.75	4.79

Fonte: o Autor(2020)

Observando os resultados mostrados na Tabela 22, podemos observar que os modelos não apresentam diferença significativa entre si, menos de 72.73% de diferença entre os erros em cada um dos modelos avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de todos os modelos avaliados, assim como ocorreu para os resultados mostrados na Tabela

21, se basearem em árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

Com base nas análses do modelos estimadores do uso da CPU, podemos notar que o erro é menor que 5.5% de uso de CPU para Galaxy A10 e 4.0% para o Moto G6, resultando em um erro bastante baixo quando comparado à variabilidade dos dados que é de 24% para Galaxy A10 e 8% para o Moto G6.

Após realizarmos as análises dos erros dos modelos gerados usando as cargas de trabalho mostradas na Tabela 5, avaliamos os erros cometidos pelos novos modelos gerados a partir da detecção do *drift* de dados, conforme discutido na Seção 7.7. Na próxima seção, avaliamos os erros cometidos pelos modelos estimadores da potência instantânea a partir do novo conjunto de dados.

8.3 AVALIAÇÃO DOS MODELOS DE POTÊNCIA PARA CONJUNTO DE DADOS COM MAPAS E DEEZER

Conforme discutido na Seção 5.7, a implantação do modelo não é o final do processo de construção de nenhum sistema baseado em modelos de aprendizagem de máquina. Para avaliar uma das principais causas de degradação da capacidade preditiva dos modelos, nós implementamos em nosso módulo de monitoramento de modelos a detecção de *drift* de dados. Na Seção 7.7, nós discutimos a respeito dos experimentos realizados para avaliar a detecção de *drift* de dados e a sinalização ao módulo de criação de modelos para treinar um novo modelo.

Nesta seção, iremos avaliar os novos modelos estimadores da potência instantânea para os smartphones Galaxy A10 e Moto G6 seguindo o mesmo procedimento de análise feito na Seção 8.1 usando as cargas de trabalho mostradas nas Tabelas 5 e 9. Os resultados obtidos usando o conjunto de testes para cada smartphone avaliado, Galaxy A10 e Moto G6, são mostrados nas Tabelas 25 e 26.

Para mensurar o quão bons esses resultados de erro dos estimadores gerados pelo nosso módulo são, precisamos analisar as estatísticas do conjunto de dados de testes para os smartphones Galaxy A10 e Moto G6. Os resultados são mostrados na Tabela 23 em mW.

Analisando os dados mostrados na Tabela 23, percebemos que, embora os valores dos quantis empíricos tenham subido, houve uma diminuição na variabilidade dos dados de potência do Galaxy A10 de 522.97 mW para 512.95 mW. Esse mesmo fato não ocorreu com os dados de potência instantânea do smartphone Moto G6.

Tabela 23 – Estatísticas do conjunto de dados de teste para potência instantânea(mW)-Com Deezer Music e Google Maps

Smartphone	25° Percentil	Mediana	75° Percentil	IIQ
Galaxy A10	1012.32	1250.72	1525.27	512.95
Moto G6	434.65	1371.78	1831.70	1397.05

Fonte: o Autor(2022)

Seguindo o mesmo procedimento adotado na Seção 8.1, obtivemos os tempos necessários para realizar todo o procedmento descrito na Seção 5.3. Os resultados obtidos são mostrados na Tabela 24.

Tabela 24 – Tempos para construção dos modelos de potência por cenário-Deezer e Google Maps

Cenário	Tempo(s)	
	Galaxy A10	Moto G6
I	5426	16506
II	3238	8171
III	2288	3017
IV	706	583

Fonte: o Autor(2020)

Analisando os dados mostrados na Tabela 24, percebemos que houve uma redução drástica e consistente do tempo necessário para otimizar os hiperparâmetros, treinar e testar os modelos no cenário **IV** no qual fazemos seleção de atributos usando a técnica de eliminação recursiva de atributos. Nos outros cenários em que aplicamos seleção de atributos, houve variação na redução do tempo de geração de modelos para os dados de um smartphone mas não para o outro.

Analisando os resultados mostrados na Tabela 25, podemos observar três características relevantes:

- Os modelos mais bem classificados em cada um dos cenários estudados não apresentam diferença significativa, menos de 41.01% de diferença entre os erros em cada um dos cenários avaliados, de acordo com o procedimento descrito na Seção 5.1.4.
- Em todos os cenários, os melhores modelos elencados são obtidos a partir do algoritmo LGBM.
- O cenário em que o modelo LGBM apresentou o menor erro foi o **IV** em que também houve uma grande redução no tempo de geração do modelo, conforme mostado na Tabela 24.

Tabela 25 – Modelos para estimar potência instantânea(mW)- Erro dos modelos-Galaxy A10

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**LGBM**	239.03	242.64	241.99	243.32
	Bagging	239.96	241.05	241.47	242.25
	HistGradient Boosting	245.86	248.13	247.68	248.62
	XGB	272.56	277.64	276.70	278.57
	Gradient Boosting	420.72	426.25	425.40	427.10
II	**LGBM**	238.47	241.53	241.00	242.05
	HistGradient Boosting	251.04	254.55	253.99	255.16
	Bagging	255.18	257.88	257.35	258.37
	XGB	290.57	296.65	295.73	297.50
	Gradient Boosting	424.08	427.07	426.45	427.67
III	**LGBM**	239.65	241.83	241.40	242.25
	Bagging	241.33	243.99	243.45	244.50
	HistGradient Boosting	249.75	252.01	251.58	252.42
	XGB	303.52	312.14	310.29	313.96
	Gradient Boosting	434.27	439.81	439.03	440.65
IV	**LGBM**	237.63	241.36	240.77	241.93
	Lasso Lars IC	262.95	265.97	265.46	266.43
	Linear Regression	263.27	1.04×10^12	3.57×10^11	1.90×10^12
	XGB	268.92	275.19	274.29	276.08
	Gradient Boosting	424.60	429.41	428.73	430.17

Fonte: o Autor(2022)

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de, em todos os cenários avaliados, haver uma predominância dos modelos baseados em *ensembles* de árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

A partir dos resultados observados, verficando que o modelo gerado no cenário **IV** possui o

menor erro e também há um menor tempo de geração, o nosso módulo de criação de modelos fornece o modelo de aprendizagem de máquina gerado nesse cenário para realizar as predições de potência instantânea do Galaxy A10.

Tabela 26 – Modelos para estimar potência instantânea(mW)- Erro dos modelos-Moto G6

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**Decision Tree**	133.19	142.27	140.24	144.56
	Extra Tree	135.68	143.30	141.65	145.11
	Bagging	300.74	305.49	304.84	306.21
	Extra Trees	374.16	377.39	376.75	377.98
II	**Extra Tree**	102.10	109.73	108.36	111.06
	Decision Tree	107.08	113.98	112.90	115.10
	Bagging	263.47	266.13	265.60	266.70
	Extra Trees	523.61	573.79	567.15	580.27
III	**Decision Tree**	99.37	110.72	109.33	112.03
	Extra Tree	101.07	109.84	108.34	111.36
	Bagging	259.94	263.11	262.61	263.66
	Extra Trees	378.77	382.96	382.30	383.66
IV	**Decision Tree**	101.12	109.80	108.47	111.00
	Extra Tree	105.86	112.89	111.70	113.91
	Extra Trees	390.67	400.21	398.48	402.11

Fonte: o Autor(2022)

Ao analisarmos os resultados mostrados na Tabela 26, percebemos que, assim como ocorreu com os modelos estimadores da potência instantânea para o Galaxy A10, houve um pequeno aumento no erro cometido pelos modelos gerados para estimar a potência instantânea do smartphone Moto G6. Também podemos observar que, com exceção do cenário **II** em que fazemos seleção de atributos usando a técnica de filtragem com informação mútua, em todos os outros cenários avaliados há uma predominância do modelos gerados a partir do algoritmo **Decision Tree**.

Novamente, utilizando o procedimento descrito na Seção 5.3, não obervamos diferença significativa entre os erros do modelos uma vez que não há uma diferença superior a 101.8%.

Dessa forma, novamente o nosso módulo de geração de modelos escolhe o modelo gerado no cenário **IV** para estimar a potência instantânea do smartphone Moto G6.

Analisando os erros do modelos responsáveis por estimar a potência instantânea do smartphone Galaxy A10 e comparando com os modelos gerados quando estávamos considerando apenas a carga de trabalho mostrada na Tabela 5, percebemos que houve um aumento do erro cometido pelo modelo escolhido de 235.86 mW para 237.63 mW, novamente atingindo o nosso objetivo de estimar a potência instantânea a partir do padrão de comportamento do usuário. Na próxima seção, avaliaremos os modelos gerados pelo nosso módulo de criação de modelos proposto para estimar temperatura e o uso da CPU, considerando os novos conjuntos de dados.

8.4 AVALIAÇÃO DOS MODELOS DE USO E TEMPERATURA DA CPU COM MAPAS E DEEZER

Após a construção dos modelos para estimar a potência instantânea do smartphone usando o novo conjunto de dados, conforme discutido na Seção 7.3, nós precisamos construir modelos para estimar o uso e a tempertaura da CPU, usando o novo conjunto de dados construído usando as cargas de trabalho mostradas nas Tabelas 5 e 9.

Os resultados obtidos pelos modelos responsáveis por estimar a temperatura da CPU usando o conjunto de Testes para cada smartphone avaliado, Galaxy A10 e Moto G6, são mostrados nas Tabelas 29 e 30. Nessas tabelas, nós podemos: a) avaliar a média dos erros dos 30 modelos gerados para cada algoritmo avaliado usando a métrica RMSE e b) classificar os erros para escolher o modelo com o menor erro. Todos os erros são expresos em $^{\circ}C$.

Para mensurar o quão bons esses resultados de erro dos estimadores gerados pelo nosso módulo são, precisamos analisar as estatísticas do conjunto de dados de testes para os smartphones Galaxy A10 e Moto G6. Os resultados são mostrados na Tabela 27 em $^{\circ}C$.

Tabela 27 – Estatísticas do conjunto de dados de teste para temperatura da CPU($^{\circ}C$)-Com Deezer Music e Google Maps

Smartphone	25° Percentil	Mediana	75° Percentil	IIQ
Galaxy A10	38.00	40.00	42.00	4.00
Moto G6	31.00	34.70	36.50	5.50

Fonte: o Autor(2022)

Seguindo o mesmo procedimento adotado na Seção 8.1, obtivemos os tempos necessários

para realizar todo o procedmento descrito na Seção 5.3. Os resultados obtidos são mostrados na Tabela 28.

Tabela 28 – Tempos para construção dos modelos de temperatura da CPU por cenário-Deezer e Google Maps

Cenário	Tempo(s)	
	Galaxy A10	Moto G6
I	4868	26856
II	4841	6061
III	2420	6364
IV	972	4760

Fonte: o Autor(2020)

Analisando os dados mostrados na Tabela 28, percebemos que houve uma redução do tempo necessário para otimizar os hiperparâmetros, treinar e testar os modelos nos cenários **III** e **IV**, nos quais fazemos seleção de atributos usando as técnicas de algoritmos genéticos e eliminação recursiva de atributos, respectivamente.

Analisando os resultados mostrados na Tabela 29, podemos observar duas características relevantes:

- Os modelos mais bem classificados não apresentam diferença significativa em relação aos outros modelos, menos de 10.0% de diferença entre os erros em cada um dos cenários avaliados, de acordo com o procedimento descrito na Seção 5.1.4.
- Em três dos quatro cenários avaliados, os melhores modelos elencados são obtidos a partir do algoritmo LGBM.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de, em todos os cenários avaliados, novamente, haver uma predominância dos modelos baseados em *ensembles* de árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

A partir dos resultados observados, como a diferença entre os melhores modelos em cada um dos cenários avaliados não foi considerada significativa, o módulo de geração de modelos escolhe o melhor modelo do cenário com o menor tempo, o cenário **IV** na análse em questão.

Observando os resultados mostrados na Tabela 19, podemos observar três características relevantes:

Tabela 29 – Modelos para estimar Temperatura da CPU($^{\circ}C$)- Erro dos modelos-Galaxy A10

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**Bagging**	0.80	0.83	0.83	0.84
	LGBM	0.81	0.83	0.83	0.83
	Extra Trees	1.03	1.06	1.05	1.06
	HistGradient Boosting	1.44	1.49	1.48	1.49
	XGB	1.64	1.68	1.68	1.69
II	**LGBM**	0.77	0.79	0.79	0.79
	Bagging	0.81	0.83	0.83	0.84
	HistGradient Boosting	0.92	0.93	0.93	0.94
	Extra Trees	1.02	1.06	1.05	1.06
	XGB	1.13	1.18	1.18	1.19
III	**LGBM**	0.78	0.81	0.80	0.81
	Bagging	0.80	0.84	0.83	0.84
	HistGradient Boosting	0.87	0.89	0.89	0.89
	Extra Trees	1.02	1.05	1.05	1.06
	XGB	1.67	1.72	1.71	1.73
IV	**LGBM**	0.79	0.80	0.80	0.80
	Decision Tree	0.86	0.88	0.88	0.89
	Extra Trees	1.01	1.06	1.05	1.06
	XGB	2.14	2.24	2.22	2.25
	Gradient Boosting	3.40	3.43	3.42	3.44

Fonte: o Autor(2022)

- Em três dos quatro cenários avaliados, os modelos elencados com o menor erro possuem o mesmo erro, 0.05%.
- O modelo mais bem classificado no cenário **I** apresenta diferença significativa em relação aos outros modelos, mais de 15.85% de diferença entre os erros em cada um dos cenários avaliados, de acordo com o procedimento descrito na Seção 5.1.4.
- Embora os modelos baseados no algoritmo **Decision Tree** não sejam elencados como

Tabela 30 – Modelos para estimar Temperatura da CPU($^{\circ}C$)- Erro dos modelos-Moto G6

Cenário	Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
I	**Extra Tree**	0.07	0.08	0.08	0.08
	Decision Tree	0.07	0.08	0.08	0.08
	Bagging	0.20	0.21	0.20	0.21
	Extra Trees	0.99	1.06	1.04	1.07
II	**Decision Tree**	0.05	0.05	0.05	0.05
	Extra Tree	0.05	0.05	0.05	0.05
	Bagging	0.15	0.16	0.16	0.16
	Extra Trees	1.30	1.43	1.40	1.46
III	**Decision Tree**	0.05	0.05	0.05	0.05
	Extra Tree	0.05	0.06	0.06	0.06
	Bagging	0.14	0.14	0.14	0.14
	Extra Trees	1.09	1.16	1.15	1.17
IV	**Extra Tree**	0.05	0.05	0.05	0.06
	Decision Tree	0.05	0.06	0.06	0.06
	Extra Trees	1.01	1.10	1.09	1.12

Fonte: o Autor(2022)

o melhor modelo em todos os cenários, naqueles em que ele aparece classificado em segundo colocado, eles possuem o mesmo erro dos modelos classificados em primeiro lugar.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de em todos os cenários avaliados, assim como ocorreu com os resultados mostrados na Tabela 29, haver uma predominância dos modelos baseados em árvores de regressão. Esse resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

A partir dos resultados observados, como a diferença entre os melhores modelos em três dos quatro cenários avaliados não foi considerada significativa, o módulo de geração de modelos escolhe o melhor modelo do cenário com o menor tempo, o cenário **IV** na análse em questão.

Com base nas análises do modelos estimadores da temperatura da CPU, podemos notar

que o erro é menor que $1°C$, resultando em um erro bastante baixo quando comparado à variabilidade dos dados, $4.00°C$ para o Galaxy A10 e $5.50°C$ para o Moto G6.

Em seguida realizamos a análise dos erros cometidos pelos modelos responsáveis por estimar o uso da CPU baseado no padrão de comportamento do usuário usando o novo conjunto de dados. Para construir os modelos estimadores do uso da CPU, como fizemos na Seção 8.2, nós não avaliamos o erro cometido em cenários com seleção de atributos pois selecionamos os atributos preditores, conforme discutido na Seção 7.3. Dessa forma, os erros cometidos pelos estimadores do uso da CPU são mostrados nas Tabelas 32 e 33. As estatísticas do conjunto de testes usado para avaliar o erro dos modelos são mostradas na Tabela 31.

Tabela 31 – Estatísticas do conjunto de dados de teste para uso da CPU(%))-Com Deezer Music e Google Maps

Smartphone	25° Percentil	Mediana	75° Percentil	IIQ
Galaxy A10	37.00	48.00	53.00	16.00
Moto G6	5.00	12.00	14.00	9.00

Fonte: o Autor(2022)

Analisando os dados mostrados na Tabela 31, percebemos que, embora os valores dos quantis empíricos tenham subido, houve uma diminuição na variabilidade dos dados de uso da CPU do Galaxy A10 de 24.00% para 16.00% de uso de CPU. Esse mesmo fato não ocorreu com os dados de uso de CPU do smartphone Moto G6.

Tabela 32 – Modelos para estimar Uso da CPU(%)- Erro dos modelos-Galaxy A10

Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
LGBM	5.92	6.13	6.10	6.15
Extra Tree	5.92	6.15	6.11	6.19
Bagging	6.02	6.17	6.15	6.20
Extra Trees	6.44	6.60	6.57	6.64
XGB	6.68	6.81	6.79	6.83

Fonte: o Autor(2020)

Observando os resultados mostrados na Tabela 32, podemos observar que os modelos não apresentam diferença significativa entre si, menos de 30.00% de diferença entre os erros em cada um dos modelos avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

Outra característica importante a ser destacada a partir da análise dos resultados é o fato de, todos os modelos avaliados, novamente, se basearem em árvores de regressão. Esse

resultado se deve ao fato de, como discutido na Seção 8.1, os algoritmos baseados em árvores terem grande poder de generalização.

Tabela 33 – Modelos para estimar Uso da CPU(%)- Erro dos modelos-Moto G6

Algoritmo	Erro do Melhor Modelo	Média dos Erros	IC_Inferior dos Erros	IC_Superior dos Erros
Decision Tree	3.95	4.03	4.02	4.05
Extra Tree	3.95	4.06	4.04	4.08
Bagging	3.97	4.06	4.05	4.08
Extra Trees	4.02	4.14	4.12	4.16
XGB	5.67	5.96	5.93	5.99

Fonte: o Autor(2020)

Observando os resultados mostrados na Tabela 33, podemos observar que os modelos não apresentam diferença significativa entre si, menos de 75.00% de diferença entre os erros em cada um dos modelos avaliados, de acordo com o procedimento descrito na Seção 5.1.4.

Com base nas análises dos modelos estimadores do uso da CPU, podemos notar que o erro é menor que 6.00% de uso de CPU para Galaxy A10 e 4.0% para o Moto G6, resultando em um erro bastante baixo quando comparado à variabilidade dos dados que é de 16% para Galaxy A10 e 9% para o Moto G6.

8.5 CONSIDERAÇÕES FINAIS

Os resultados apresentados em nossos estudos de caso mostraram uma grande predominância dos algoritmos baseados em árvores e *ensembles* de árvores de regressão para extrair o padrão de comportamento do usuário e relacionar esse padrão de comportamento ao uso dos dispositivos que compõem o smartphone. Esses resultados reforçaram a ideia de que, embora tenhamos um grande avanço na área de aprendizagem de máquina com as redes neurais de aprendizagem profunda, os modelos baseados em árvores ainda são muito úteis por terem grande poder de generalização e um baixo custo computacional, fato esse mostrado pelos tempos necessários para criar os modelos. Após a análise feita dos nossos módulos de software responsáveis por tratar os dados, criar os modelos baseados no padrão de comportamento do usuário e monitorá-los, concluímos os objetivos definidos para a realização desta pesquisa.

Com os resultados obtidos e discutidos neste capítulo, pudemos averiguar que construímos módulos de software capazes de tratar os dados de uso produzidos pelo nosso aplicativo de

monitoramento e criar modelos capazes de estimar o uso dos dispositivos que compõem os smartphones baseados em um conjunto de preditores passados em tempo de execução sem a necessidade de refazer os módulos para que novos modelos possam ser construídos.

Foi possível perceber, através da realização dos experimentos descritos neste capítulo, que utilizamos vários regressores e os resultados são relevantes e consistentes em termos do erro alcançado. Com algoritmos de aprendizagem de máquina simples e rápidos de serem ajustados, treinados e testados, conseguimos modelar uso dos dispositivos que compõem o smartphone baseados no padrão de comportamento do usuário.

9 CONCLUSÃO E TRABALHOS FUTUROS

A utilização de smartphones para a realização das tarefas diárias vem crescendo fortemente na última década, partindo de uma participação de mercado de apenas $8,04\%$ em 2011 para $56,9\%$ em 2021. Com esse crescimento, surge um grande desafio para os fabricantes desses dispositivos móveis e desenvolvedores de aplicações: reduzir o consumo energético visando aumentar a capacidade computacional sem aumentar a demanda por energia ao mesmo tempo.

Desde a popularização do uso de smartphones, muitas pesquisas vêm sendo realizadas com o objetivo de modelar ou otimizar o consumo energético desses smartphones, usando: técnicas de engenharia de software que relacionam a chamada de funções ao consumo energético; otimizações no funcionamento dos dispositivos que compõem os smartphones diminuindo a potência de operação deles e, agora mais recentemente, modelando o padrão de comportamento do usuário.

A presente tese de doutorado estende a modelagem do padrão de comportamento do usuário e consumo energético dos smartphones, apresentando, primeiramente, um procedimento metodológico para relacionar o padrão comportamento de uso com a utilização dos dispositivos que compõem o smartphone, além de relacionar o uso dos dispositivos que compõem o smartphone ao consumo energético dele com uma técnica de análise pouco intrusiva.

Em seguida, ao longo desta pesquisa, propomos um conjunto de módulos de software capazes de implementar o procedimento metodológico proposto utilizando uma estratégia de criação de modelos de aprendizagem de máquina de forma flexível de modo a permitir estabelecer os atributos de entrada e a variável objetivo em tempo de execução e, com isso, possibilitando a criação de diversos modelos de aprendizagem de máquina capazes de relacionar o padrão de comportamento ao uso dos dispositivos que compõem o smartphone.

Antes de apresentar o procedimento metodológico discutido no Capítulo 5 para relacionar o padrão de comportamento ao uso dos dispositivos que compõem o smartphone, precisamos criar o *workflow* da solução, dicutido no Capítulo 4. O *workflow* da solução discutido anteriormente teve por finalidade nos possibilitar uma visão geral a respeito da solução do problema de pesquisa discutido na Seção 1.2: conceber modelos de aprendizagem de máquina capazes de relacionar o padrão de comportamento do usuário com o uso dos dispositivos que compõem o smartphone.

Os experimentos descritos na Seção 7.4 nos permitiram entender que era possível criar

modelos simples do ponto de vista computacional capazes de estabelecer uma relação entre o padrão de comportamento do usuário e o uso dos dispositivos que compõem o smartphone com uma quantidade de dados não muito superior a cinco horas de coleta. Já os experimentos descritos na Seção 7.5, nos mostrou que – embora não tívessemos acesso às estruturas de funcionamento internas do Adroid – era possível estabelecer atributos relevantes para os modelos de aprendizagem de máquina criados pelos módulos de software propostos nesta pesquisa.

Durante esta pesquisa, através dos experimentos descritos na Seção 7.6, pudemos perceber que – embora os modelos de aprendizagem de máquina sejam fortemente influenciados pela qualidade dos dados usados para treiná-los – os nossos módulos de software propostos nesta pesquisa eram capazes de construir modelos com um erro 18.47% menor quando comparado a um modelo que possui um comportamento semelhante ao exibido pelo dispositivo responsável por gerar esses dados enviesados.

Os experimentos realizados e descritos no Capítulo 8 nos mostraram que a utilização de mais de uma técnica de seleção de atributos com o objetivo de reduzir o tempo de otimização de hiperparâmetros, treinamento e teste dos modelos selecionados foi uma estratégia bastante acertada. Pudemos perceber durante a realização e análise dos experimentos que a estratégia de seleção de atributos mais adequada varia de acordo com o dispositivo e com o smartphone que estão sendo modelados.

Para atingir o objetivo geral proposto nesta pesquisa, não precisávamos gerar mais que um modelo a partir da classificação feita pelo método de avaliação de modelos descritos na Seção 5.1.4. No entanto, ao gerarmos mais de um modelo seguindo o procedimento descrito para avaliar e classificar os modelos, temos os requisitos necessários para avançarmos em trabalhos futuros, realizando a extração do conhecimento para fazer recomendações.

A realização desta pesquisa nos mostrou que conseguimos resolver o problema de modelar a relação existente entre o padrão de comportamento do usuário e o uso dos dispositivos que compõem o smartphone fazendo considerações a respeito de diversos fatores como:

- Condições de uso do smartphone e carga de trabalho;
- Variáveis como intensidade de brilho da tela e tipo de conexão de internet e
- Ambientes de hardware e software como a versão do Android utilizada e a acurácia do medidor de bateria utilizado.

Do ponto de vista de aprendizagem de máquina, conseguimos mostrar que com algoritmos de aprendizagem de máquina pouco complexos conseguimos capturar a relação entre o comportamento do usuário e o uso dos dispositivos que compõem o smartphone.

9.1 LIMITAÇÕES E AMEAÇAS À VALIDAÇÃO

Durante esta pesquisa, optamos por construir módulos de software e realizar experimentos não intrusivos do ponto de vista do sistema operacional. Essa abordagem de trabalho nos permitiu construir um aplicativo de monitoramento de uso do usuário que pode ser usado em diversos smartphones com versões distintas do sistema operacional Android. No entanto, a validade dos dados obtidos e, consequentemente, a criação dos modelos são ameaçadas pelo estado interno do sistema operacional Android.

Durante nossos experimentos para mensurar a sobrecarga energética gerada pelo nosso aplicativo de monitoramento, nós pudemos perceber que, mesmo quando colocado em "modo avião", em que boa parte dos dispositivos que compõem o smartphone são desativados, obtivemos resultados que flutuam bastante, indicando uma sobrecarga em alguns testes e não influenciando o consumo energético em outros. Essas flutuações nos motivaram a investigar a respeito da acurácia dos dispositivos medidores de bateria e, com isso, incluir em nosso procedimento metodológico um método para avaliar os modelos construídos a partir de conjuntos de dados enviesados.

Outra limitação presente em nossa pesquisa é o fato de não termos realizado testes com vários usuários devido ao tempo que tínhamos para investigar tudo o que foi levantado durante a criação do nosso *workflow* descrito no Capítulo 4. Para modelar o comportamento do usuário, um passo importante para indicar aos fabricantes dos smartphones e desenvolvedores de aplicação o que mais impacta o consumo energético do smartphone, precisávamos realizar testes com vários usuários a fim de fazer a extração do conhecimento. Dessa forma, conseguiríamos fazer recomendações a esses *stakeholders*.

No estado atual em que se encontra a nossa pesquisa, não somos capazes de aferir o aumento do consumo energético causado pela constante utilização dos modelos de aprendizagem de máquina criados. Não conseguimos mensurar essa sobrecarga energética causada pelo uso dos modelos de aprendizagem criados porque, no estado atual de nossa pesquisa, não conseguimos implantar os modelos gerados no smartphone.

Ao identificarmos essas limitações presentes em nossa pesquisa, somos capazes de iden-

tificar as próximas contribuições que precisam ser realizadas a fim de podermos substituir a estratégia de ajustes utilizada pelo Android na atualidade que não considera o padrão de comportamento do usuário por uma mais inteligente e responsiva, a fim de aumentar o tempo de uso dos smartphones que utilizem os modelos criados.

9.2 TRABALHOS FUTUROS

Embora a presente pesquisa relatada nesta tese tenha alcançado uma série de avanços relacionados à modelagem de uso de dispositivos a partir do padrão de comportamento do usuário, há ainda uma série de oportunidades no que tange à criação de mecansimos de otimização e redução energética dos smartphones.

Utilizando a API de comunicação com os módulos de software propostos nesta pesquisa, uma pesquisa maior – envolvendo mais usuários e modelos de smartphones – pode ser conduzida a fim de extrair conhecimento permitindo a fabricantes de smartphones e desenvolvedores de aplicações criarem otimizadores que avaliem a configuração dos dispositivos e o respectivo consumo energético sem a necessidade de implantar as soluções em smartphones físicos. Além disso, pode ser realizada uma extensão desta pesquisa que leve em consideração a estratégia proposta por Xia *et al.* (2020) para construir um modelo comum a usuários com padrões de uso semelhantes, a fim de melhorar a capacidade preditiva dos modelos de uso gerados pelos módulos de software propostos na presença de dados escassos.

Outro avanço importante, fruto desta pesquisa, é o desenvolvimento de uma nova estratégia que possa executar os algoritmos inteligentes no próprio smartphone. Essa estratégia permitirá aos fabricantes substituírem o mecanismo atual, dependente do arquivo *power_profile.xml* por uma estratégia inteligente, mais adequada ao perfil de uso do usuário.

Tomando por base a estrutura modular proposta nesta pesquisa , é possível adicionar mais estratégias de otimização e escolha de modelos inteligentes de modo a possibilitar a inserção de estratégias multi-objetivo, possibilitando aos fabricantes e aos desenvolvedores de aplicações definir outros objetivos além da redução do erro do modelo.

Outra abordagem interessante proposta por Farias *et al.* (2020), é adotar uma divisão dos dados baseada em suas respectivas similaridades. Os autores apontam uma melhoria na acurácia de predição consistente em 75% das bases de dados analisadas. Podemos, dessa forma, em trabalhos futuros, adaptar a estratégia proposta por eles para algoritmos de regressão e incorporá-la em nosso módulo de análise de dados de modo a avaliar se, com ela, construímos

melhores modelos.

Durante a realização desta pesquisa, ao avaliarmos as diversas técnicas de aprendizagem de máquina, optamos por utilizar algoritmos de aprendizagem de máquina *shallow* por eles terem um tempo de treinamento e estimação menor que as abordagens que utilizam redes neurais profundas. No entanto, podemos aplicar técnicas como a defendida por Farias *et al.* (2022) em um trabalho recentemente disponibilizado. Nele, Farias *et al.* (2022) propõem um procedimento que permite o treinamento de diversas redes neurais multicamadas independentes com diferentes números de neurônios e funções de ativação, explorando o princípio da localidade e as capacidades de paralelização das CPUs e GPUs modernas.

Adotando estratégias como a proposta por Farias *et al.* (2022), podemos avaliar modelos que utilizem algoritmos de aprendizagem de máquina e redes neurais de forma empilhada. Usando essa abordagem, utilizaríamos os algoritmos de aprendizagem de máquina baseados em árvores para selecionar atributos e redes neurais para estimar o uso de dispositivos.

Ao realizarmos experimentos com diversos usuários, poderemos fazer uma outra derivação desta pesquisa utilizando os 5 melhores modelos elencados pelo nosso módulo de criação de modelos para fazer a extração de conhecimento e, utilizando ferramentas como SHAPE, podermos fazer recomendações aos desenvolvedores e fabricantes de smartphones.

Enfim, com a publicação dos artigos e da própria tese, a comunidade poderá fazer extensões propondo outras abordagens que se baseiem no perfil de uso do usuário de modo a proporcionar uma experiência de uso mais personalizada.

REFERÊNCIAS

AKIBA, T.; SANO, S.; YANASE, T.; OHTA, T.; KOYAMA, M. Optuna: A next-generation hyperparameter optimization framework. In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.: s.n.], 2019.

ALAWNAH, S.; SAGAHYROON, A. Modeling of smartphones' power using neural networks. *EURASIP Journal on Embedded Systems*, Springer, v. 2017, n. 1, p. 22, 2017.

AMR, T. *Generalized Linear Models (GLM)*. 2022. Acesso em: Out./2022. Disponível em: <https://bit.ly/3MjksIN>.

ARENA, G. *Samsung Galaxy S20*. 2021. Acesso em: Jul./2021. Disponível em: <https://bit.ly/3iAzn44>.

ARENA, G. *Samsung Galaxy S21*. 2021. Acesso em: Jul./2021. Disponível em: <https://bit.ly/3fONUrn>.

ATHUKORALA, K.; LAGERSPETZ, E.; KÜGELGEN, M. von; JYLHÄ, A.; OLINER, A. J.; TARKOMA, S.; JACUCCI, G. How carat affects user behavior: implications for mobile battery awareness applications. In: ACM. *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. [S.l.], 2014. p. 1029–1038.

AUTHORITY, A. *Fact check: Is smartphone battery capacity growing or staying the same?* 2021. Acesso em: Jul./2021. Disponível em: <https://bit.ly/2TEk2G7>.

BANKO, M.; BRILL, E. Scaling to very very large corpora for natural language disambiguation. In: *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*. [S.l.: s.n.], 2001. p. 26–33.

BETTINI, C.; CIVITARESE, G.; PRESOTTO, R. Caviar: Context-driven active and incremental activity recognition. *Knowledge-Based Systems*, Elsevier, p. 105816, 2020.

BONETTO, A.; FERRONI, M.; MATTEO, D.; NACCI, A.; MAZZUCCHELLI, M.; SCIUTO, D.; SANTAMBROGIO, M. Mpower: Towards an adaptive power management system for mobile devices. In: *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*. ACM, 2012. p. 318–325. Disponível em: <http://doi.acm.org/10.1145/1669112.1669135>. Acesso em: 19.1.2014.

BRODOWSKI, D. *CPUFreq Governors*. 2022. Acesso em: Out./2022. Disponível em: <https://bit.ly/3Ts4r5Y>.

BROWNLEE, J. *Master Machine Learning Algorithms: Discover How They Work and Implement Them From Scratch*. [S.l.]: Jason Brownlee, 2016.

BROWNLEE, J. *Probability for Machine Learning: Discover How To Harness Uncertainty With Python*. [S.l.]: Machine Learning Mastery, 2019.

BURKOV, A. *Machine Learning Engineering*. [S.l.]: True Positive Incorporated, 2020. ISBN 9781999579579.

CALZOLARI, M. *Sklearn Genetic*. 2022. Acesso em: Nov./2022. Disponível em: <https://bit.ly/3TZ9RFX>.

CHEN, A.; SEN, P. K. Advancement in battery technology: A state-of-the-art review. In: IEEE. *2016 IEEE Industry Applications Society Annual Meeting*. [S.l.], 2016. p. 1–10.

CHEN, T.; GUESTRIN, C. XGBoost: A scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2016. (KDD '16), p. 785–794. ISBN 978-1-4503-4232-2.

ÇIÇEK, E.; GÖREN, S. Smartphone power management based on convlstm model. *Neural Computing and Applications*, Springer, v. 33, n. 13, p. 8017–8029, 2021.

COMUNITY, P. *Python Data Analysis Library*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/38cTrSn>.

DAI, P.; HO, S.-S.; RUDZICZ, F. Sequential behavior prediction based on hybrid similarity and cross-user activity transfer. *Knowledge-Based Systems*, Elsevier, v. 77, p. 29–39, 2015.

DEVELOPERS, A. *BatteryManager*. 2020. Acesso em: Jan./2020. Disponível em: <https://bit.ly/2MYKcgb>.

DEVELOPERS, A. *Broadcasts*. 2022. Acesso em: Jul./2022. Disponível em: <https://goo.gl/JRdTi5>.

DEVELOPERS, A. *Perfil de uso da bateria com o Batterystats e o Battery Historian*. 2022. Acesso em: Mai./2022. Disponível em: <https://bit.ly/3MeDPS1>.

DEVELOPERS, A. *Services*. 2022. Acesso em: Jul./2022. Disponível em: <https://goo.gl/wm3atb>.

DING, M.; WANG, T.; WANG, X. Establishing smartphone user behavior model based on energy consumption data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, ACM New York, NY, v. 16, n. 2, p. 1–40, 2021.

DOCKER. *Overview of Docker Compose*. 2022. Acesso em: Jul./2022. Disponível em: <https://dockr.ly/3o6dueV>.

EFRON, B.; TIBSHIRANI, R. *An Introduction to the Bootstrap*. [S.l.]: Taylor & Francis, 1994. (Chapman & Hall/CRC Monographs on Statistics & Applied Probability). ISBN 9780412042317.

EIBEN, A.; SMITH, J. *Introduction to Evolutionary Computing*. [S.l.]: Springer Berlin Heidelberg, 2015. (Natural Computing Series). ISBN 9783662448748.

ELLIOTT, J.; KOR, A.; OMOTOSHO, O. A. Energy consumption in smartphones: an investigation of battery and energy consumption of media related applications on android smartphones. 2017.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. F. d. *Inteligência artificial: uma abordagem de aprendizado de máquina*. [S.l.]: LTC, 2021.

FARIAS, F.; LUDERMIR, T.; BASTOS-FILHO, C. Similarity based stratified splitting: an approach to train better classifiers. *arXiv preprint arXiv:2010.06099*, 2020.

FARIAS, F. C.; LUDERMIR, T. B.; BASTOS-FILHO, C. J. A. Embarrassingly parallel independent training of multi-layer perceptrons with heterogeneous architectures. *arXiv preprint arXiv:2206.08369*, 2022.

FISHER, R. *The Design of Experiments*. [S.l.]: Oliver and Boyd, 1935. (The Design of Experiments).

GALLI, S.; SAMIULLAH, C.; GALLI, N. *Feature-engine:A Python library for Feature Engineering and Selection*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3AIiBaQ>.

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, 2019. ISBN 9781492032595.

GIFT, N.; DEZA, A. *Practical MLOps*. [S.l.]: O'Reilly Media, 2021. ISBN 9781098102982.

GOOGLE. *Battery Historian*. 2017. Acesso em: Mai./2022. Disponível em: <https://bit.ly/3MeDPS1>.

HANDYORG. *Mzip-Android*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/35TAq5B>.

HAO, S.; LI, D.; HALFOND, W. G.; GOVINDAN, R. Estimating mobile application energy consumption using program analysis. In: IEEE PRESS. *Proceedings of the 2013 International Conference on Software Engineering*. [S.l.], 2013. p. 92–101.

HAUSTANT, A. *Flask-RESTPlus*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3o8Eb3i>.

HUYEN, C. *Designing Machine Learning Systems*. [S.l.]: O'Reilly Media, 2022. ISBN 9781098107918.

INC, J. A. *Android Shell*. 2021. Acesso em: Jun./2021. Disponível em: <https://bit.ly/30pgx5n>.

KAMATH, U.; LIU, J. *Explainable Artificial Intelligence: An Introduction to Interpretable Machine Learning*. [S.l.]: Springer International Publishing, 2021. ISBN 9783030833558.

KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W.; MA, W.; YE, Q.; LIU, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, v. 30, p. 3146–3154, 2017.

KERNEL, T. L. *CPU Performance Scaling*. 2019. Acesso em: Jan./2019. Disponível em: <https://bit.ly/2sdxOln>.

KERNEL, T. L. *Generic Thermal Sysfs driver How To*. 2019. Acesso em: Jan./2020. Disponível em: <https://bit.ly/2FEWlmt>.

KIM, Y.; PARTERNA, F.; TILAK, S.; ROSING, T. S. Smartphone analysis and optimization based on user activity recognition. In: IEEE. *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*. [S.l.], 2015. p. 605–612.

KOPEC, D.; SAFARI, a. O. M. C. *Classic Computer Science Problems in Python Video Edition*. [S.l.]: Manning Publications, 2019.

KRAUS, M. *Using Bayesian Optimization to reduce the time spent on hyperparameter tuning*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/38nkGK5>.

KUHN, M.; JOHNSON, K. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. [S.l.]: CRC Press, 2019. (Chapman & Hall/CRC Data Science Series). ISBN 9781351609463.

KUMAR, S. *7 Hyperparameter Optimization Techniques Every Data Scientist Should Know*. 2022. Acesso em: Out./2022. Disponível em: <https://bit.ly/3RR0k1U>.

LAROSE, D. *Discovering Knowledge in Data: An Introduction to Data Mining*. [S.l.]: Wiley, 2014. (Wiley Series on Methods and Applications in Data Mining). ISBN 9781118873571.

LI, H.; LIU, X.; MEI, Q. Predicting smartphone battery life based on comprehensive and real-time usage data. *arXiv preprint arXiv:1801.04069*, 2018.

LI, M.; LU, J.; CHEN, Z.; AMINE, K. 30 years of lithium-ion batteries. *Advanced Materials*, Wiley Online Library, v. 30, n. 33, p. 1800561, 2018.

LINUX, A. *Backlight*. 2020. Acesso em: Jan./2020. Disponível em: <https://bit.ly/35BdMis>.

MEHROTRA, D.; SRIVASTAVA, R.; NAGPAL, R.; NAGPAL, D. Multiclass classification of mobile applications as per energy consumption. *Journal of King Saud University-Computer and Information Sciences*, Elsevier, v. 33, n. 6, p. 719–727, 2021.

MITCHELL, T. *Machine Learning*. [S.l.]: McGraw-Hill Education, 1997. (McGraw-Hill international editions - computer science series). ISBN 9780070428072.

MORE, P. *Motorola Moto G6 XT1925-3*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/2Rj46nt>.

NG, A. Y. et al. Preventing"overfitting"of cross-validation data. In: CITESEER. *ICML*. [S.l.], 1997. v. 97, p. 245–253.

NUCCI, D. D.; PALOMBA, F.; PROTA, A.; PANICHELLA, A.; ZAIDMAN, A.; LUCIA, A. D. Software-based energy profiling of android apps: Simple, efficient and reliable? In: IEEE. *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*. [S.l.], 2017. p. 103–114.

PAGE, L. man. *Iostat(1)*. 2019. Acesso em: Jan./2019. Disponível em: <https://bit.ly/2Nhd5od>.

PAGE, L. man. *vmstat(8)*. 2020. Acesso em: Jan./2020. Disponível em: <https://bit.ly/2NeCyi4>.

PANDALA, S. R.; SILVA, B. B. *Lazy Predict*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/340gVNS>.

PARADISO, J. A.; STARNER, T. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive computing*, Ieee, v. 4, n. 1, p. 18–27, 2005.

PATHAK, A.; HU, Y. C.; ZHANG, M.; BAHL, P.; WANG, Y.-M. Fine-grained power modeling for smartphones using system call tracing. In: ACM. *Proceedings of the sixth conference on Computer systems*. [S.l.], 2011. p. 153–168.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PRAMANIK, P. K. D.; SINHABABU, N.; MUKHERJEE, B.; PADMANABAN, S.; MAITY, A.; UPADHYAYA, B. K.; HOLM-NIELSEN, J. B.; CHOUDHURY, P. Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage. *IEEE Access*, IEEE, v. 7, p. 182113–182172, 2019.

PRECHELT, L. et al. Proben1: A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, 1994.

PROJECT, A. *Power Profiles for Android*. 2022. Acesso em: Out./2022. Disponível em: <https://bit.ly/3dWzd7T>.

PROJECT, A. O. S. *Power Profiles for Android*. 2022. Acesso em: Mai./2022. Disponível em: <https://bit.ly/3x9blVv>.

PROJECTS, P. *Flask Web Development*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/36XJ1Wv>.

RAJ, E. *Engineering MLOps: Rapidly build, test, and manage production-ready machine learning life cycles at scale*. [S.l.]: Packt Publishing, 2021. ISBN 9781800566323.

ROMANSKY, S.; BORLE, N. C.; CHOWDHURY, S.; HINDLE, A.; GREINER, R. Deep green: Modelling time-series of software energy consumption. In: IEEE. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.], 2017. p. 273–283.

ROMANYSHYN, Y. *Deepchecks:Open Source package for ML Validation*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3uhYe3f>.

SARKER, I. H.; ABUSHARK, Y. B.; KHAN, A. I. Contextpca: Predicting context-aware smartphone apps usage based on machine learning techniques. *Symmetry*, Multidisciplinary Digital Publishing Institute, v. 12, n. 4, p. 499, 2020.

SCIKIT-LEARN. *Feature ranking with recursive feature elimination*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3xt9l88>.

SCIKIT-LEARN. *Linear Models*. 2022. Acesso em: Jul./2022. Disponível em: <https://bit.ly/3IHpHAv>.

SCIKIT-LEARN. *LocalOutlierFactor*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/32SrjTf>.

SCIKIT-LEARN. *Mean Squared Error*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3kUAXjd>.

SCIKIT-LEARN. *MinMaxScaler*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3GgWhGq>.

SCIKIT-LEARN. *OrdinalEncoder*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/2Vhh8HX>.

SCIKIT-LEARN. *QuantileTransformer*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3ocXvwb>.

SCIKIT-LEARN. *Select features according to the k highest scores*. 2022. Acesso em: Nov./2022. Disponível em: <https://bit.ly/2w0hp1Q>.

SHAFIK, R. A.; YANG, S.; DAS, A.; MAEDA-NUNEZ, L. A.; MERRETT, G. V.; AL-HASHIMI, B. M. Learning transfer-based adaptive energy minimization in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, IEEE, v. 35, n. 6, p. 877–890, 2015.

SHEARER, F. *Power Management in Mobile Devices*. [S.l.]: Elsevier Science, 2011. ISBN 9780080556406.

SHYE, A.; SCHOLBROCK, B.; MEMIK, G. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In: IEEE. *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. [S.l.], 2009. p. 168–178.

SOUCH. *AndroidCPU*. 2020. Acesso em: Jan./2020. Disponível em: <https://bit.ly/36GAs20>.

SPECS, P. *pickle — Python object serialization*. 2022. Acesso em: Jan./2022. Disponível em: <https://bit.ly/3IOZa2V>.

STATISTA. *Number of smartphone subscriptions worldwide from 2016 to 2026 (in millions)*. 2021. Acesso em: Jul./2021. Disponível em: <https://bit.ly/3BSMtBm>.

STATS, S. G. *Desktop vs Mobile Market Share Worldwide*. 2021. Acesso em: Jul./2021. Disponível em: <https://bit.ly/3i9jPUN>.

STUDIOS, Q. *opencv-android*. 2020. Acesso em: Jan./2020. Disponível em: <https://bit.ly/2szQEDC>.

TANNOR, P. *Data Drift vs. Concept Drift: What Are the Main Differences?* 2022. Acesso em: Jul./2022. Disponível em: <https://bit.ly/3aPkaew>.

TARKOMA, S.; SIEKKINEN, M.; LAGERSPETZ, E.; XIAO, Y. *Smartphone Energy Consumption: Modeling and Optimization*. [S.l.]: Cambridge University Press, 2014. (Smartphone Energy Consumption: Modeling and Optimization). ISBN 9781107042339.

TEACHING, L. K. *Linux Device Model*. 2022. Acesso em: Out./2022. Disponível em: <https://bit.ly/3yvm6Sj>.

TIAN, Z.; WANG, Z.; LI, H.; YANG, P.; MAEDA, R. K. V.; XU, J. Multi-device collaborative management through knowledge sharing. In: IEEE. *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. [S.l.], 2018. p. 22–27.

VALÉRIO, R. *Foreground App Checker for android*. 2020. Acesso em: Jan./2020. Disponível em: <https://bit.ly/39YMDJH>.

WADE, C.; GLYNN, K. *Hands-On Gradient Boosting with XGBoost and scikit-learn: Perform accessible machine learning and extreme gradient boosting with Python*. [S.l.]: Packt Publishing, 2020. ISBN 9781839213809.

WANG, C.; YAN, F.; GUO, Y.; CHEN, X. Power estimation for mobile applications with profile-driven battery traces. In: IEEE PRESS. *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*. [S.l.], 2013. p. 120–125.

WAZLAWICK, R. S. *Metodologia de pesquisa para ciência da computação*. Terceira edição. Rio de Janeiro: LTC, 2022.

XIA, T.; LI, Y.; FENG, J.; JIN, D.; ZHANG, Q.; LUO, H.; LIAO, Q. Deepapp: predicting personalized smartphone app usage via context-aware multi-task learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM New York, NY, USA, v. 11, n. 6, p. 1–12, 2020.

XIE, T.; ZHENG, Q.; ZHANG, W. Recognizing physical contexts of mobile video learners via smartphone sensors. *Knowledge-Based Systems*, Elsevier, v. 136, p. 75–84, 2017.

YAN, K.; TAN, J.; FU, X. Improving energy efficiency of mobile devices by characterizing and exploring user behaviors. *Journal of Systems Architecture*, Elsevier, v. 98, p. 126–134, 2019.

YE, A. *The Beauty of Bayesian Optimization, Explained in Simple Terms*. 2022. Acesso em: Out./2022. Disponível em: <https://bit.ly/3rlth5E>.

YOON, C.; KIM, D.; JUNG, W.; KANG, C.; CHA, H. AppScope: Application energy metering framework for android smartphone using kernel activity monitoring. In: *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX Association, 2012. p. 387–400. ISBN 978-931971-93-5. Disponível em: <https://www.usenix.org/conference/atc12/technical-sessions/presentation/yoon>.

ZHENG, A. *Evaluating Machine Learning Models: A Beginner's Guide to Key Concepts and Pitfalls*. [S.l.]: O'Reilly Media, 2015. ISBN 9781491932469.

ZHENG, A.; CASARI, A. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. [S.l.]: O'Reilly Media, 2018. ISBN 9781491953198.

APÊNDICE A – ARQUIVO DE PERFIS DE POTÊNCIA

Aqui será mostrado, na listagem 2, o arquivo modelo para os perfis de potência fornecido por PROJECT(2022b).

Código Fonte 2 – Modelo de arquivo de perfis de potência

```
<?xml version="1.0" encoding="utf-8"?>
<!--
**
** Copyright 2009, The Android Open Source Project
**
** Licensed under the Apache License, Version 2.0 (the "License")
** you may not use this file except in compliance with the License.
** You may obtain a copy of the License at
**
**     http://www.apache.org/licenses/LICENSE-2.0
**
** Unless required by applicable law or agreed to in writing, software
** distributed under the License is distributed on an "AS IS" BASIS,
** WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
** See the License for the specific language governing permissions and
** limitations under the License.
*/
-->
<device name="Android">
  <!-- Most values are the incremental current used by a feature,
       in mA (measured at nominal voltage).
       The default values are deliberately incorrect values.
       OEMs must measure and provide actual values before
       shipping a device.
       Example real-world values are given in comments, but they
       are totally dependent on the platform and can vary
       significantly, so should be measured on the shipping platform
       with a power meter. -->
  <!-- Display related values. -->
  <!-- Average battery current draw of display0 while in ambient mode, including
      backlight.
       There must be one of these for each display, labeled:
       ambient.on.display0, ambient.on.display1, etc...
       Each display suffix number should match it is ordinal in its display
           device config.
  -->
  <item name="ambient.on.display0">0.1</item>  <!-- ~100mA -->
  <!-- Average battery current draw of display0 while on without backlight.
       There must be one of these for each display, labeled:
       screen.on.display0, screen.on.display1, etc...
```

```
39        Each display suffix number should match it is ordinal in its display
              device config.
    -->
41  <item name="screen.on.display0">0.1</item>  <!-- ~100mA -->
    <!-- Average battery current draw of the backlight at full brightness.
43        The full current draw of display N at full brightness should be the sum of
               screen.on.displayN
          and screen.full.displayN
45        There must be one of these for each display, labeled:
          screen.full.display0, screen.full.display1, etc...
47        Each display suffix number should match it is ordinal in its display
              device config.
    -->
49  <item name="screen.full.display0">0.1</item>  <!-- ~100mA -->
    <item name="bluetooth.active">0.1</item> <!-- Bluetooth data transfer, ~10mA --
        >
51  <item name="bluetooth.on">0.1</item>  <!-- Bluetooth on & connectable, but not
        connected, ~0.1mA -->
    <item name="wifi.on">0.1</item>  <!-- ~3mA -->
53  <item name="wifi.active">0.1</item>  <!-- WIFI data transfer, ~200mA -->
    <item name="wifi.scan">0.1</item>  <!-- WIFI network scanning, ~100mA -->
55  <item name="audio">0.1</item> <!-- ~10mA -->
    <item name="video">0.1</item> <!-- ~50mA -->
57  <item name="camera.flashlight">0.1</item> <!-- Avg. power for camera flash,
        ~160mA -->
    <item name="camera.avg">0.1</item> <!-- Avg. power use of camera in standard
        usecases, ~550mA -->
59  <item name="gps.on">0.1</item> <!-- ~50mA -->
    <!-- Radio related values. For modems without energy reporting support in
        firmware, use
61        radio.active, radio.scanning, and radio.on. -->
    <item name="radio.active">0.1</item> <!-- ~200mA -->
63  <item name="radio.scanning">0.1</item> <!-- cellular radio scanning for signal,
         ~10mA -->
    <!-- Current consumed by the radio at different signal strengths, when paging
        -->
65  <array name="radio.on"> <!-- Strength 0 to BINS-1 -->
        <value>0.2</value> <!-- ~2mA -->
67      <value>0.1</value> <!-- ~1mA -->
    </array>
69  <!-- Additional power consumption by CPU excluding cluster and core when
         running -->
71  <array name="cpu.active">
        <value>0.1</value>
73  </array>
    <!-- A list of heterogeneous CPU clusters, where the value for each cluster
        represents the
```

```
75        number of CPU cores for that cluster.
          Ex:
77        <array name="cpu.clusters.cores">
            <value>4</value> // cluster 0 has cpu0, cpu1, cpu2, cpu3
79          <value>2</value> // cluster 1 has cpu4, cpu5
          </array> -->
81    <array name="cpu.clusters.cores">
        <value>1</value> <!-- cluster 0 has cpu0 -->
83    </array>
      <!-- Different CPU speeds for cluster 0 as reported in
85        /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state.
          There must be one of these for each cluster, labeled:
87        cpu.speeds.cluster0, cpu.speeds.cluster1, etc... -->
      <array name="cpu.speeds.cluster0">
89      <value>400000</value> <!-- 400 MHz CPU speed -->
      </array>
91    <!-- Current at each CPU speed for cluster 0, as per 'cpu.speeds.cluster0'.
          Like cpu.speeds.cluster0, there must be one of these present for
93        each heterogeneous CPU cluster. -->
      <array name="cpu.active.cluster0">
95      <value>0.1</value>  <!-- ~100mA -->
      </array>
97    <!-- Current when CPU is idle -->
      <item name="cpu.idle">0.1</item>
99    <!-- Memory bandwidth power values in mA at the rail. There must be one value
          for each bucket defined in the device tree. -->
101   <array name="memory.bandwidths">
        <value>22.7</value> <!-- mA for bucket: 100mb/s-1.5 GB/s memory bandwidth -->
103   </array>
      <!-- This is the battery capacity in mAh (measured at nominal voltage) -->
105   <item name="battery.capacity">1000</item>
      <!-- Wifi related values. -->
107   <!-- Idle Receive current for wifi radio in mA. 0 by default-->
      <item name="wifi.controller.idle">0</item>
109   <!-- Rx current for wifi radio in mA. 0 by default-->
      <item name="wifi.controller.rx">0</item>
111   <!-- Tx current for wifi radio in mA. 0 by default-->
      <item name="wifi.controller.tx">0</item>
113   <!-- Current at each of the wifi Tx levels in mA. The number of tx levels
          varies per device
          and is available only of wifi chipsets which support the tx level
              reporting. Use
115         wifi.tx for other chipsets. none by default -->
      <array name="wifi.controller.tx_levels"> <!-- mA -->
117   </array>
      <!-- Operating volatage for wifi radio in mV. 0 by default-->
119   <item name="wifi.controller.voltage">0</item>
```

```
  <array name="wifi.batchedscan"> <!-- mA -->
    <value>.0002</value> <!-- 1-8/hr -->
    <value>.002</value>  <!-- 9-64/hr -->
    <value>.02</value>   <!-- 65-512/hr -->
    <value>.2</value>    <!-- 513-4,096/hr -->
    <value>2</value>    <!-- 4097-/hr -->
  </array>
  <!-- Cellular modem related values.-->
  <modem>
    <!-- Modem sleep drain current value in mA. -->
    <sleep>0</sleep>
    <!-- Modem idle drain current value in mA. -->
    <idle>0</idle>
    <!-- Modem active drain current values.
         Multiple <active /> can be defined to specify current drain for
             different modes of
         operation.
         Available attributes:
             rat - Specify the current drain for a Radio Access Technology.
                   Available options are "LTE", "NR" and "DEFAULT".
                   <active rat="default" /> will be used for any usage that does
                       not match any other
                   defined <active /> rat.
             nrFrequency - Specify the current drain for a frequency level while
                 NR is active.
                           Available options are "LOW", "MID", "HIGH", "MMWAVE",
                               and "DEFAULT",
                           where,
                           "LOW" indicated <1GHz frequencies,
                           "MID" indicates 1GHz to 3GHz frequencies,
                           "HIGH" indicates 3GHz to 6GHz frequencies,
                           "MMWAVE"indicates >6GHz frequencies.
                           <active rat="NR" nrFrequency="default"/> will be used
                               for any usage that
                           does not match any other defined <active rat="NR" />
                               nrFrequency.
    -->
    <active rat="DEFAULT">
      <!-- Transmit current drain in mA. -->
      <receive>0</receive>
      <!-- Transmit current drains in mA. Must be defined for all levels (0 to 4)
           -->
      <transmit level="0">0</transmit>
      <transmit level="1">0</transmit>
      <transmit level="2">0</transmit>
      <transmit level="3">0</transmit>
      <transmit level="4">0</transmit>
```

```
    </active>
    <!-- Additional <active /> may be defined.
         Example:
             <active rat="LTE"> ... </active>
             <active rat="NR" nrFrequency="MMWAVE"> ... </active>
             <active rat="NR" nrFrequency="DEFAULT"> ... </active>
    -->
  </modem>
  <item name="modem.controller.voltage">0</item>
  <!-- GPS related values. Default is 0.-->
  <array name="gps.signalqualitybased"> <!-- Strength 0 to 1 -->
    <value>0</value>
    <value>0</value>
  </array>
  <item name="gps.voltage">0</item>
</device>
```

Fonte: PROJECT2022a

APÊNDICE B – PROCEDIMENTO DE COLETA DOS DADOS

Com a seleção dos atributos dos dispositivos que compõem o smartphone, conforme descrito na Seção 5.1.2, nós pudemos projetar e estabelecer um procedimento de coleta utilizando as APIs do Android para coletar os atributos selecionados dos dispositivos e as características de estado do sistema operacional inerentes ao uso do usuário.

As *intents* **ACTION_SCREEN_OFF** e **ACTION_SCREEN_ON** são os mecanismos utilizados pelo Android para prover a informação necessária a respeito do estado, ligado ou desligado, da tela.

A classe **WifiManager**, presente no Android, é responsável por prover informações a respeito do estado, ligado ou desligado, do WiFi e sobre a intensidade de sinal do Wifi. Já a classe **BluetoothAdapter** provê informações a respeito do estado do Bluetooth. A classe **AndroidConnectionType** nos possibilita obter a informação necessária para atualizar o estado, conectado e desconectado, da conexão de Rádio (3G/4G).

Para monitorar a intensidade de sinal da rede móvel, nós utilizamos a classe **PhoneStateListener**. Essa classe possui o método *onSignalStrengthChanged* responsável por capturar as mudanças na intensidade de sinal da rede móvel.

A classe **NetworkStat** do Android é usada para prover informações a respeito dos dados trafegados pelas redes, WiFi e 3G/4G. Para obter informações a respeito da orientação da tela (vertical/horizontal), nós criamos uma especialização da classe *OrientationEventListener* usando o método *onOrientationChanged*.

Após buscarmos uma forma de obtermos a temperatura da CPU utilizando as APIs providas pelo Android, não obtivemos sucesso. Diante dessa constatação, fomos novamente ao *kernel* do Linux, sobre o qual o Android funciona, e verificamos que ele nos fornece essa informação através da estrutura de diretório */sysfs*.

Conforme discutimos na Seção 5.1, a estrutura de diretórios */sysfs* é o meio pelo qual o *kernel* disponibiliza, no espaço do usuário, o seu modelo de dispositivos. Para buscar informações a respeito da temperatura da CPU, utilizamos o arquivo */sys/class/thermal/thermal_zone0/temp*, o qual representa o driver Thermal SysFs (KERNEL, 2019b).

Para mensurar a frequência de cada CPU do smartphone, utilizamos o arquivo virtual *cpuinfo_cur_freq* do CPU Performance Scaling (KERNEL, 2019a).

A ferramenta do Linux IoStat (PAGE, 2019) é usada para obter informações a respeito da

quantidade de Kilobytes lidos e escritos no disco. Já a ferramenta Vmstat (PAGE, 2020) foi usada para obter informações a respeito da quantidade de Swap In, Swap Out e chaveamentos de contexto.

A classe de sistema Backlight do Linux (LINUX, 2020) é responsável por mensurar o brilho da tela e prover essa informação para o nosso serviço desenvolvido. Para obter o uso de CPU, nós utilizamos a classe CPUInfo do aplicativo AndroidCPU (SOUCH, 2020). A média e o desvio padrão da intensidade dos píxels coloridos da imagem na tela são providos pela bibiloteca opencv-android (STUDIOS, 2020).

Para obter o aplicativo que está em *foreground*, a cada momento em que os dados de uso são registrados, nós utilizamos o *Foreground App Checker for Android* (VALÉRIO, 2020), que é responsável por monitorar as aplicações em *foreground* nas versões mais recentes do Android.

Além dos atributos monitorados descritos acima, nós incluímos no processo de caracterização de uso a corrente instantânea (ma) e a tensão(v) usando o procedimento descrito na Seção 5.1.1.