UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

VINÍCIUS RODRIGUES FRANDOLOSO

ANÁLISE DE DADOS PARA IDENTIFICAÇÃO DE ATIVIDADES DE ESTACIONAMENTO DE VEÍCULOS POR MEIO DE APRENDIZADO DE MÁQUINA

PATO BRANCO

2023

VINÍCIUS RODRIGUES FRANDOLOSO

ANÁLISE DE DADOS PARA IDENTIFICAÇÃO DE ATIVIDADES DE ESTACIONAMENTO DE VEÍCULOS POR MEIO DE APRENDIZADO DE MÁQUINA

Data Analysis for Identification of Vehicle Parking Activities through Machine Learning

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR), como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Dr. Érick Oliveira Rodrigues

PATO BRANCO

2023

VINÍCIUS RODRIGUES FRANDOLOSO

ANÁLISE DE DADOS PARA IDENTIFICAÇÃO DE ATIVIDADES DE ESTACIONAMENTO DE VEÍCULOS POR MEIO DE APRENDIZADO DE MÁQUINA

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de Aprovação: 16 de junho de 2023.

Profa. Dra. Soelaine Rodrigues Ascari
Universidade Tecnológica Federal do Paraná

Prof. Dr. Fábio Favarim
Universidade Tecnológica Federal do Paraná

Profa. Dra. Eliane Maria De Bortoli Favero
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2023

RESUMO

A dificuldade de lembrar onde um veículo foi estacionado é um problema comum, principalmente em ambientes desconhecidos. Este estudo teve como objetivo desenvolver uma análise e estudo a respeito da possibilidade de distinguir, por meio de algoritmos de aprendizagem de máquina, as atividades de estacionar e estar em movimento de um veículo. Um aplicativo móvel foi desenvolvido utilizando o *framework* Flutter com a finalidade de coletar dados por meio de sensores, como o velocímetro, giroscópio e magnetômetro, para treinar algoritmos de aprendizado de máquina. Foram empregados os algoritmos k-NN, Random Forest e SVM para analisar os dados coletados. A pesquisa foi conduzida em duas fases, utilizando conjuntos de dados distintos para treinamento e testes. O desempenho dos algoritmos foi avaliado com base na acurácia alcançada na classificação. Em resumo, os resultados revelaram que o algoritmo Random Forest obteve a melhor eficácia na classificação do estacionamento de veículos, alcançando uma acurácia de 92,71% na primeira fase e 94,12% na segunda fase. Esses resultados destacam a capacidade do sistema proposto em distinguir com precisão as atividades de estacionamento em diferentes contextos.

Palavras-chave: acurácia; aprendizado de máquina; flutter.

ABSTRACT

The difficulty of remembering where a vehicle was parked is a common problem, especially in unfamiliar environments. This study aimed to develop an analysis and investigation regarding the possibility of distinguishing parking activities and vehicle motion through machine learning algorithms. A mobile application was developed using the Flutter framework to collect data through sensors such as the speedometer, gyroscope, and magnetometer, in order to train machine learning algorithms. The k-NN, Random Forest, and SVM algorithms were employed to analyze the collected data. The research was conducted in two phases, using distinct datasets for training and testing. The performance of the algorithms was evaluated based on the accuracy achieved in classification. In summary, the results revealed that the Random Forest algorithm achieved the highest efficacy in classifying vehicle parking, reaching an accuracy of 92.71% in the first phase and 94.12% in the second phase. These results highlight the capability of the proposed system to accurately distinguish parking activities in different contexts.

Keywords: accuracy; machine learning; flutter.

LISTA DE ALGORITMOS

LISTA DE CÓDIGOS-FONTE

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

SIGLAS

AM	Aprendizagem de Máquina
AOT	*Ahead-of-time*
API	*Application Programming Interface*
ART	*Android Runtime*
ASF	*Apache Software Foundation*
CPU	*Central Processing Unit*
DEX	*Dalvik Executable*
DOD	*Department of Defense*
DOT	*Department of Transportation*
GC	*Garbage Collection*
GPS	*Global Positioning System*
HAL	*Hardware Abstract Layers*
HTC	*High Tech Computer Corporation*
JIT	*Just-In-Time*
k-NN	*K-Nearest Neighbors*
LG	*Lucky Goldstar*
NASA	*National Aeronautics and Space Administration*
NAVSAT	*Navy Navigation Satellite System*
NRL	*United States Naval Research Laboratory*
OHA	*Open Handset Alliance*
PVT	Posição, Velocidade e Tempo
RBF	*Radial Basis Function*
SDK	*Software Development Kit*
SO	Sistema Operacional
SVC	*Support Vector Classifier*
SVM	*Support Vector Machine*
TXT	*Text*
UTFPR	Universidade Tecnológica Federal do Paraná
WEKA	*Waikato Environment for Knowledge Analysis*

SUMÁRIO

1 INTRODUÇÃO

O ser humano possui uma grande capacidade de reconhecimento de padrões e identificação de objetos, entretanto tarefas simples de serem realizadas por seres humanos, se tornam desafiadoras para computadores e dispositivos móveis. Por conta disso, cientistas se esforçam para ajudar os computadores a lidar com certas categorias de problemas, utilizando exemplos oferecidos por meio de metodologias específicas, ao invés de desenvolver algoritmos individuais para resolver essas dificuldades.

Essa abordagem é conhecida como aprendizado de máquina. Definido por Mitchell e Mitchell (1997) como: “A capacidade de melhorar o desempenho na realização de alguma tarefa por meio da experiência”.

Segundo Basavaraju e Varde (2017) até o momento, os sistemas de computação móveis, mostraram a partir de estudos realizados pelo mesmo, que as abordagens de aprendizado de máquina funcionam bem em seus sistemas. Visto que possuem sensores prontamente disponíveis, como câmeras, giroscópios, *Global Positioning System* (GPS) e acelerômetro, o que auxilia os desenvolvedores de aplicativos a criarem diversos serviços unindo os sensores com a aprendizagem de máquina.

Um aplicativo para dispositivos móveis é um programa de software feito especificamente para ser executado em eletrônicos portáteis, como smartphones e tablets. A combinação de mídia, tecnologia da informação, Internet e tecnologias de ponta deu origem aos aplicativos móveis (GAO *et al.*, 2015).

Novos serviços são adicionados continuamente aos dispositivos móveis. Visto isso, os usuários regulares passam a utilizar esses serviços que estão prontamente disponíveis em seus equipamentos eletrônicos, seja para navegação online, acesso a redes sociais, localização de outros usuários ou na utilização de outros serviços (LECHETA, 2015).

Assim, esta pesquisa busca desenvolver e avaliar um sistema de classificação que auxilie na identificação da localização do veículo estacionado, oferecendo uma solução prática e útil para um problema comum do dia a dia.

1.1 OBJETIVOS

Análise e estudo a respeito da possibilidade de distinguir, por meio de algoritmos de aprendizado de máquina, as atividades de estacionar e estar em movimento de um veículo, com

base em dados de velocidade, giroscópio e magnetômetro obtidos em tempo real por meio de um aplicativo móvel.

1.1.1 Objetivos Específicos

- Coletar dados de velocidade, giroscópio e magnetômetro em tempo real utilizando um aplicativo móvel desenvolvido para esse fim.
- Realizar atividades de pré-processamento nos dados coletados, incluindo normalização, exclusão e adição de colunas.
- Avaliar o desempenho dos modelos de classificação obtidos utilizando métricas de avaliação, como por exemplo a acurácia.

1.2 JUSTIFICATIVA

Neste trabalho será necessário realizar uma análise de um sistema de classificação para distinguir as atividades relacionadas ao estacionamento de veículos. Utilizando dados coletados por sensores como velocímetro, giroscópio e magnetômetro, e aplicando algoritmos de aprendizado de máquina como k-NN, *Random Forest* e SVM, busca-se identificar padrões e características que permitam reconhecer o momento em que um veículo está sendo estacionado.

A aplicação desse sistema de classificação pode trazer benefícios significativos aos usuários, ao fornecer informações precisas sobre a localização do veículo estacionado, evitando o estresse e a frustração de não encontrá-lo. Além disso, o sistema pode ser integrado a aplicativos móveis existentes, oferecendo uma solução prática e conveniente para esse problema recorrente.

Por meio de duas fases de treinamento e testes com conjuntos de dados distintos, será possível avaliar o desempenho dos algoritmos de classificação e identificar os atributos mais relevantes na classificação das atividades de estacionamento. Essa análise contribuirá para compreender a viabilidade e eficácia do sistema proposto.

2 REFERENCIAL TEÓRICO

Neste capítulo conceitos fundamentais de Geolocalização, GPS, Aprendizado de Máquina e Sistemas Operacionais são abordados. Na seção 2.5 uma breve explicação sobre o *framework* utilizado neste trabalho é realizada. Na seção 2.6 , são abordadas técnicas de Aprendizado de Máquina e alguns algoritmos de classificação são exemplificados.

2.1 GEOLOCALIZAÇÃO

A geolocalização é a real localização geográfica de um objeto, tal qual é possível obter utilizando um radar, smartphones, ou até mesmo possuindo um computador conectado à Internet. Está relacionada e é semelhante com a definição de um posicionamento geográfico, mas se concentra em estabelecer uma localização significativa, como um endereço de rua, ao invés de um conjunto de coordenadas geográficas (GUSTAFSSON, 2012).

Segundo Gentile *et al.* (2012) as correlações geométricas entre as coordenadas dos pontos de referência junto com as medições de alcance e ângulo relacionados são usados em abordagens tradicionais de geolocalização (não baseadas em levantamento). Nesses casos, os dispositivos sem fio com informações de localização pré-programadas, ou derivadas de GPS são os pontos de referência (coordenadas X e Y).

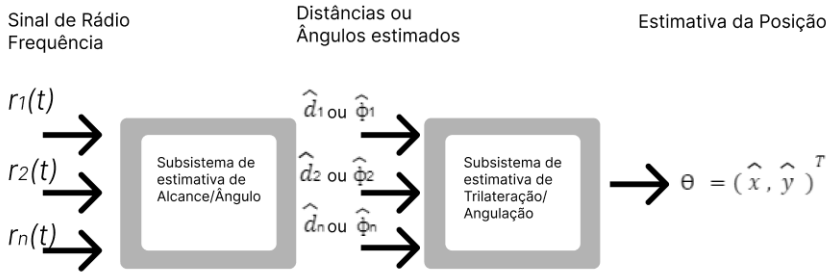
O dispositivo móvel troca sinais de rádio frequência com os pontos de referência (recebendo suas próprias informações de posição) para estimar a distância ou ângulo para cada um dos locais de referência definidos. Como o dispositivo móvel possui diversas ferramentas para medir a distância e as coordenadas do ponto de referência, pode descobrir uma posição desconhecida, usando uma variedade de estratégias (geométrica, otimização, etc.) (GENTILE *et al.*, 2012).

Segundo (GENTILE *et al.*, 2012), três principais aspectos influenciam a precisão dos dados de localização: posição dos pontos de referência, precisão das estimativas de alcance ou ângulos, e por fim, a configuração geométrica dos pontos de referência e do local desconhecido. As técnicas de geolocalização sem levantamento, utilizam duas fases para calcular a estimativa de localização: estimativa de alcance/ângulo e também pela triangulação/angulação.

Na Figura 1 é possível verificar a representação de um sistema clássico de geolocalização, as informações de alcance ou ângulo ($\hat{d_1}$ / $\hat{\phi_1}$) são extraídas dos sinais de rádio frequência ($r_1(t)$, $r_2(t), \cdots, r_n(t)$) recebidos. Então utilizando técnicas de trilateração/angulação, é possível

obter as coordenadas da posição estimada $\theta = (\hat{x}, \hat{y})^T$.

Figura 1 – Sistema clássico de geolocalização



Fonte: Fonte adaptada Gentile *et al.* (2012).

As tecnologias que utilizam geolocalização e que dependem exclusivamente de redes sem fio, como a hora de chegada, diferença de hora de chegada, ângulo de chegada e o avanço de tempo, disponibilizam um tempo para a primeira correção menor do que o GPS (DJUKNIC; RICHTON, 2001).

Segundo Djuknic e Richton (2001) os sistemas que possuem geolocalização, oferecem uma implementação rápida e capacidade de rastreamento contínuo para aplicativos de navegação sem a complexidade e os custos adicionais de substituição de aparelhos ou atualizações dos mesmos. Essas tecnologias oferecem uma oportunidade de negócio para operadores de rede, podendo se tornar provedores exclusivos de serviços, como prover as informações de localização dos assinantes.

O ponto negativo é que a geolocalização baseada em rede fornece uma menor precisão do que o GPS e também requer maiores investimentos em equipamentos para construir uma estação base. Outro fator que preocupa é a privacidade dos usuários, visto que as provedoras de serviços possuem acesso a dados privados de seus clientes (DJUKNIC; RICHTON, 2001).

2.2 NAVEGAÇÃO

A navegação é conhecida como a ciência de levar uma embarcação ou até mesmo uma pessoa de um ponto a outro. Está presente diariamente na vida das pessoas, as habilidades básicas de navegação devem ser utilizadas por exemplo, para dirigir até o trabalho ou caminhar até alguma loja. Na maioria dos casos essas habilidades dependem da visão, bom senso e pontos de referência (KAPLAN; HEGARTY, 2006).

Alguns dispositivos de navegação usam sinais eletrônicos, os quais, são mais complicados, estes podem ser chamados de radionavegação, então o usuário pode calcular sua posição

usando sinais de um ou mais dispositivos de radionavegação, seus receptores realizam cálculos relevantes, como por exemplo (alcance, direção e também hora prevista de chegada). O receptor pode manipular apenas uma parte dos sinais recebidos com os cálculos de navegação sendo realizados em outro lugar (KAPLAN; HEGARTY, 2006).

2.3 *GLOBAL POSITIONING SYSTEM* - GPS

Várias agências governamentais dos Estados Unidos, como o Departamento de Defesa (DOD), a *National Aeronautics and Space Administration* (NASA) e o Departamento de Transporte (DOT), estavam interessadas em criar sistemas de satélite para determinação de posição tridimensional no início dos anos 1960. O sistema ideal teria cobertura global, operando em todos os climas, com capacidade de suportar plataformas dinâmicas e com excelente precisão (KAPLAN; HEGARTY, 2006).

O satélite TRANSIT, também conhecido como NAVSAT (*Navy Navigation Satellite System*) foi universalmente aceito para uso em plataformas com baixa precisão, quando se tornou operacional pela primeira vez em 1964. A Marinha tentou melhorar o sistema devido a diversas limitações que o sistema possuía. Os arquitetos do satélite TRANSIT, do laboratório de Física Aplicada da Universidade Johns Hopkins apresentaram diversas variações (KAPLAN; HEGARTY, 2006).

Ao mesmo tempo, o Laboratório de Pesquisa Naval do inglês, NRL testava relógios altamente robustos baseados no espaço para permitir a transferência exata de tempo. Para este programa foi dado o nome de Timation. Os satélites de cronometragem foram modificados para adicionar uma capacidade de alcance para determinar posições bidimensionais. Para que o alcance do satélite com o usuário fosse possível, a Timation usou modulação sidetone (KAPLAN; HEGARTY, 2006).

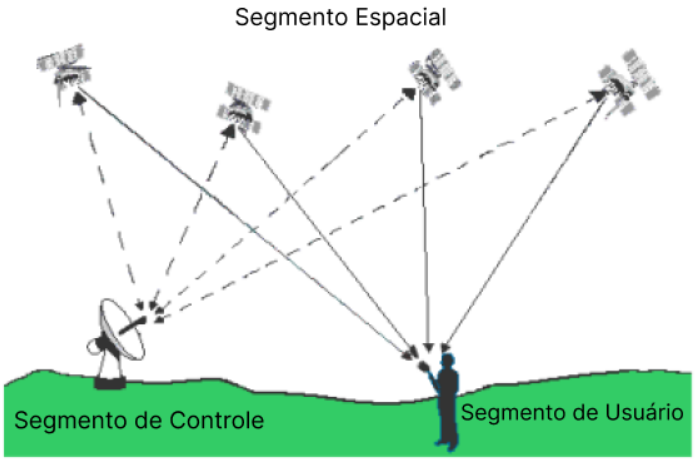
Em 1973 iniciou oficialmente o projeto do Sistema de Posicionamento Global (GPS), que é um sistema de navegação e posicionamento baseado no espaço, desenvolvido pelas Forças Armadas dos Estados Unidos com o objetivo de permitir que um soldado ou uma equipe identifique sua posição de forma independente, com uma precisão de 10 a 20 metros da posição real. O conceito de autonomia era crucial, pois era vital construir um sistema que permitisse ao soldado estabelecer sua localização sem a necessidade de qualquer outra comunicação de rádio (ALI, 2020).

O lançamento do primeiro protótipo de nave espacial foi em 1978, toda a constelação de

24 satélites tornou-se operacional em 1993, essa constelação de satélites orbitam a uma distância de onze mil milhas náuticas da Terra e seguem seis padrões orbitais diferentes. Os satélites estão em constante movimento, completando duas órbitas completas ao redor da Terra a cada 24 horas (ALI, 2020).

A Figura 2 representa o funcionamento do sistema GPS. Na figura é possível observar os três principais segmentos desse sistema.

Figura 2 – Funcionamento do Sistema GPS



Fonte: Fonte adaptada Ali (2020).

Segmento Espacial: O segmento espacial é composto pelos satélites GPS que orbitam a Terra. Esses satélites são colocados em órbita pelo Departamento de Defesa dos Estados Unidos, DOD, responsável pelo sistema GPS. A constelação de satélites garante que pelo menos quatro satélites estejam visíveis para a maioria das áreas da Terra em qualquer momento (EMBRAPA, 2020).

Segmento de Controle: O segmento de controle é composto por estações de controle terrestres e um mestre de controle. As estações de controle monitoram os satélites GPS, rastreiam suas posições e retransmitem correções e atualizações de órbita para os satélites. Essas correções são necessárias para compensar pequenas variações nas órbitas dos satélites devido a fatores gravitacionais e atmosféricos (EMBRAPA, 2020).

Segmento de Usuário: O segmento de usuário é composto pelos dispositivos receptores GPS que são usados pelos usuários para receber os sinais dos satélites e calcular sua posição, velocidade e tempo (EMBRAPA, 2020).

Os receptores GPS analisam os sinais recebidos dos satélites, incluindo os sinais da banda L, para determinar a diferença de tempo entre a transmissão do sinal pelo satélite e a

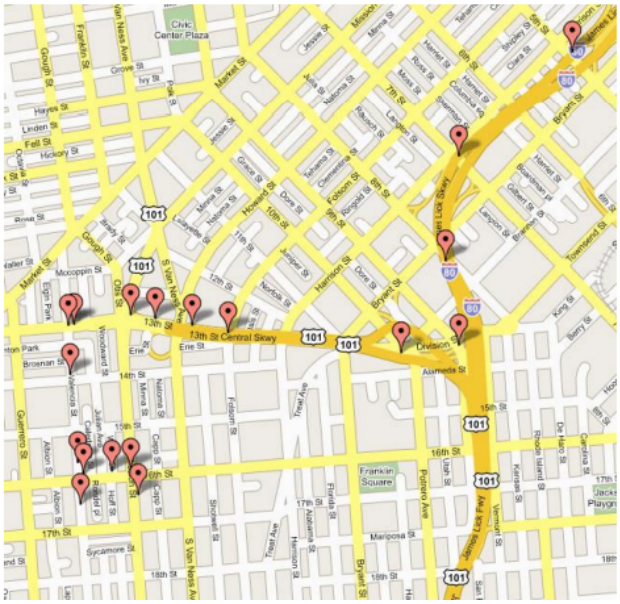
recepção pelo receptor. Com base nessas diferenças de tempo e nas informações de posição e tempo transmitidas pelos satélites, o receptor GPS realiza cálculos de triangulação para determinar sua posição geográfica (latitude e longitude), altitude e tempo (ALI, 2020).

Portanto, o segmento de usuário é o componente responsável por executar as funções de recepção e processamento dos sinais dos satélites GPS para determinar a posição, velocidade e tempo do usuário, com base na triangulação usando pelo menos quatro satélites. Os receptores GPS analisam os sinais da banda L dos satélites para identificar o PVT (Posição, Velocidade e Tempo) do usuário (ALI, 2020).

Embora a determinação do PVT seja a aplicação mais utilizada, os receptores também podem ser usados para calcular a altitude da plataforma do usuário (direção, inclinação e rotação), ou até mesmo como fonte de temporização. A função básica do GPS é a navegação tridimensional. Os receptores de navegação são projetados para aeronaves, navios e veículos terrestres, bem como para o uso pessoal. O controle geodésico e as investigações tectônicas de placas, são aplicações onde os sensores GPS, localizados em pontos de referência, fornecem correções e posicionamentos relativos (ALI, 2020).

A Figura 3 representa um mapa de pontos de posições em uma determinada área. Essa imagem foi obtido por meio de um aplicativo web onde foram coletados os dados das coordenadas das localizações escolhidas pelo usuário.

Figura 3 – Exemplo de Rotas utilizando o GPS



Fonte: (SILVA, 2014).

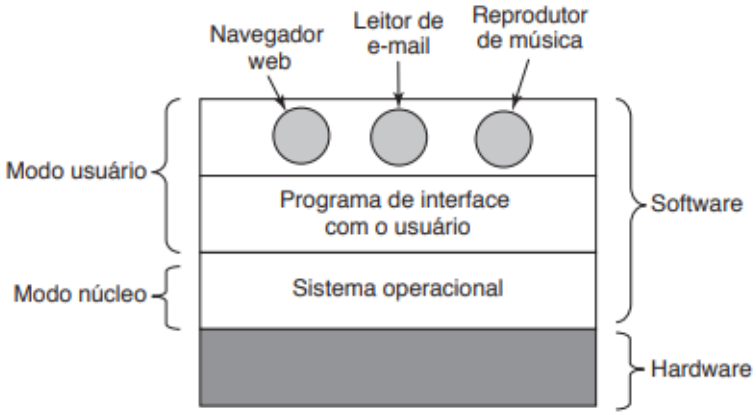
Outra aplicação relacionada com o uso do GPS, é a disseminação de tempo e frequência, que é controlada por estações de monitoramento e com base nos relógios precisos a bordo dos

satélites. Os observatórios astronômicos e instalações de telecomunicações, podem ser ajustados para sinais de tempo precisos ou controlados para frequências precisas por receptores GPS de propósito especial. Esses sinais de GPS têm sido usados em projetos de pesquisa, com o objetivo de medir características atmosféricas (ALI, 2020).

2.4 SISTEMAS OPERACIONAIS (SO)

Segundo Tanenbaum e Bos (2014), o sistema operacional é um dispositivo de software que opera em uma camada entre o software e o hardware. A Figura 4 apresenta uma representação simplificada dos principais componentes relacionados ao papel desempenhado pelo sistema operacional. O hardware é a camada inferior e é composto por componentes físicos como chips, placas, discos, teclado, monitor, entre outros. O software é a camada superior e está posicionado acima do hardware. O sistema operacional é um componente essencial do software e opera no modo núcleo, também conhecido como modo supervisor. Nesse modo, o sistema operacional possui privilégios totais de acesso aos recursos de hardware e é capaz de executar qualquer instrução suportada pela máquina. O restante do software opera no modo usuário, onde um conjunto restrito de instruções da máquina está disponível. No modo usuário, certas instruções que controlam a máquina ou realizam operações de entrada/saída são proibidas. A distinção entre o modo núcleo e o modo usuário desempenha um papel fundamental no funcionamento dos sistemas operacionais.

Figura 4 – Local onde o sistema operacional atua



Fonte: Tanenbaum e Bos (2014).

Segundo Remzi H Arpaci-Dusseau e Andrea C Arpaci-Dusseau (2018) um (SO) expõem diversas interfaces (*Application Programming Interface* - APIs), que o usuário pode chamar com o objetivo de instruir o sistema operacional sobre o que fazer e, assim, aproveitar a funcionalidade da

máquina virtual (como iniciar um programa, alocar memória, ou acessar um arquivo). Um sistema operacional é frequentemente chamado de gerenciador de recursos, porque a virtualização permite que muitos aplicativos sejam executados simultaneamente (compartilhando a *Central Processing Unit* - CPU), muitos programas acessam suas próprias instruções e dados (compartilhando memória).

Segundo Silberschatz, Galvin e Gagne (2006) os sistemas operacionais para dispositivos móveis atuais tem seus recursos que compõem o sistema, expandidos cada vez mais. O Middleware, conjunto de estruturas de software que fornecem serviços adicionais para desenvolvedores de aplicativos, é frequentemente incluído em sistemas operacionais móveis, além do kernel básico.

Cada um dos dois sistemas operacionais móveis mais populares, o iOS da Apple e o Android do Google, por exemplo, tem um núcleo central, bem como um middleware que lida com bancos de dados, multimídia e gráficos. Em resumo, o sistema operacional consiste no kernel sempre ativo, estruturas de middleware que facilitam o desenvolvimento de aplicativos e fornecem recursos e aplicativos de sistema que auxiliam no gerenciamento do sistema enquanto ele está em execução (SILBERSCHATZ; GALVIN; GAGNE, 2006).

2.4.1 *iPhone Operating System* - iOS Apple

Segundo Novac *et al.* (2017) o iOS é um sistema operacional móvel destinado para dispositivos móveis da Apple, criado pela Apple Inc. Originalmente projetado para o iPhone, este sistema operacional foi posteriormente expandido para incluir outros dispositivos da própria empresa, como o iPod, iPad e Apple TV. O iOS é semelhante ao Unix, que inclui recursos do sistema operacional Mac OS X desde a sua primeira edição.

O iOS é um dos principais fatores de sucesso para iPhones no mercado global devido à sua funcionalidade. O sistema operacional Android da Google é o principal concorrente do iOS. O sistema operacional da Apple é distinto dos seus concorrentes, devido ao fato de que a mesma empresa mantém e desenvolve tanto o sistema operacional quanto os produtos finais. Como não existe licença para instalar um sistema operacional em hardware iOS que não seja criado pela Apple, não é possível fazê-lo funcionar em outra máquina (NOVAC *et al.*, 2017).

O sistema operacional iOS possui quatro camadas: Core OS/Kernel, Core Services, Media Support e a camada Cocoa Touch, as quais serão descritas a seguir (TRACY, 2012).

Core OS/Kernel: O iOS possui um kernel conhecido como XNU e possui um núcleo Darwin, já o Core OS se aproxima do hardware. Essas APIs são escritas na linguagem de

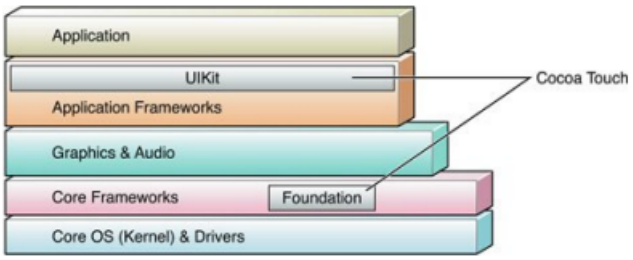
programação C e não são orientadas a objetos (NOVAC *et al.*, 2017).

Core Services: Está em um nível acima da camada base do sistema operacional, é nesse nível que é possível encontrar APIs Orientadas a Objetos. Essa camada fornece serviços básicos como manuseio de linha, administração de cobranças, conexão de rede e gerenciamento de contatos e opções. É nesses serviços que é possível utilizar recursos de hardware do dispositivo (GPS, bússola, acelerômetro e giroscópio)(NOVAC *et al.*, 2017).

Media Support: Contém as tecnologias de gráfico, áudio e vídeo. Essas tecnologias nesta camada, foram desenvolvidas a fim de tornar mais simples a implementação de aplicativos multimídia (SHEIKH *et al.*, 2013).

Cocoa Touch: É uma estrutura que permite desenvolver um aplicativo para o sistema operacional iOS, essas estruturas determinam os recursos do aplicativo. É uma interface totalmente orientada a objetos. Ela também fornece a arquitetura básica do aplicativo, incluindo suporte multitarefa, toque e notificação, como é possível observar na Figura 5 (TRACY, 2012).

Figura 5 – Distribuição da Cocoa Touch na arquitetura iOS



Fonte: Novac *et al.* (2017).

2.4.2 Sistema Operacional Android

O Android é o sistema operacional móvel do Google. É baseado no Kernel Linux e foi projetado para ser utilizado em dispositivos móveis, possui uma interface baseada em manipulação direta, usando gestos de toque que correspondem vagamente a ações do mundo real. O sucesso do Android não se deve apenas aos esforços do Google; nos bastidores do desenvolvimento da plataforma estão as principais empresas da indústria móvel, como fabricantes de celulares e provedores de serviços (LECHETA, 2016).

A Open Handset Alliance (OHA) é um grupo que auxilia no desenvolvimento da plataforma que inclui empresas como Intel, Samsung, Lucky Goldstar (LG), Motorola, Sony Ericsson, High Tech Computer Corporation (HTC), Sprint Nextel, ASUS, Acer, Dell, Garmin,

entre outras. Todo ecossistema está interessado no desenvolvimento de uma plataforma móvel poderosa e flexível com um código fonte aberto que atenda às necessidades de todos. Atualmente, o Android está disponível para uma variedade de plataformas, incluindo *smartphones* e *tablets*, televisores (*Google TV*), relógios (*Android Wear*), óculos (*Google Glass*) e automóveis (*Android Auto*), e também é o sistema operacional móvel mais popular do mundo Lecheta (2016).

O Android é a primeira plataforma de aplicativos móveis totalmente gratuita e de código aberto (*open source*), o que lhe confere uma vantagem competitiva significativa em termos de desenvolvimento, pois permite que uma ampla gama de empresas e desenvolvedores de todo o mundo contribuam para sua melhoria. Essa característica é um benefício significativo para os fabricantes de telefones móveis, pois permite que eles usem o sistema operacional Android em seus dispositivos sem precisar pagar por isso. Além disso, a licença Apache Software Foundation (ASF) permite que mudanças sejam feitas no código-fonte para criar produtos personalizados sem que essas alterações sejam compartilhadas Lecheta (2016).

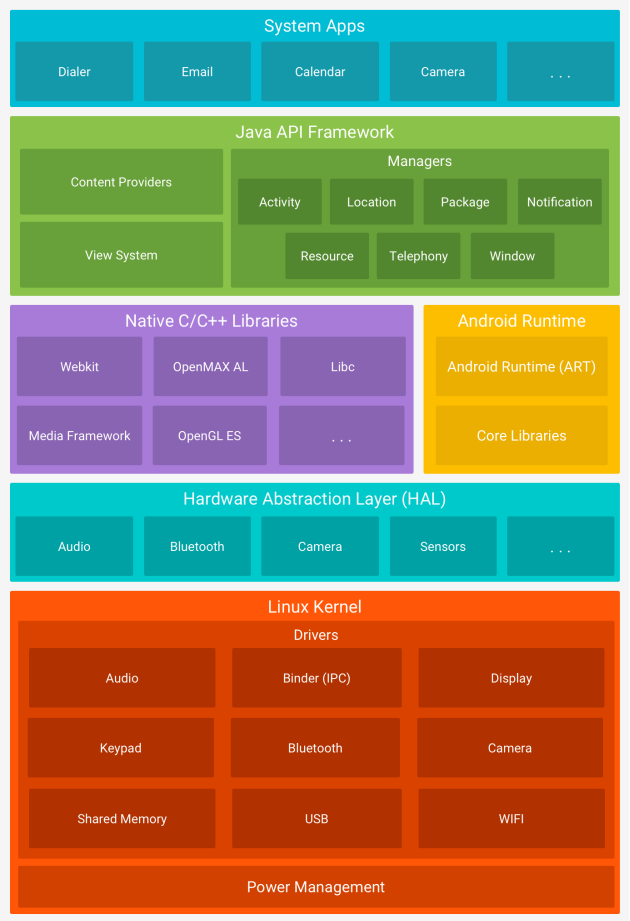
Apesar do Android ser baseado no kernel Linux, há pouco em comum com as distribuições Linux tradicionais (sendo elas em sistemas embarcados ou não), com exceção do fato de que um sistema embarcado é um sistema baseado em microprocessador no qual o computador é totalmente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Em sua forma mais simples, o Android é uma máquina virtual Java que roda no kernel Linux e fornece suporte para o desenvolvimento de aplicativos Java por meio de uma coleção de bibliotecas e serviços. Muitos dos recursos encontrados no Android, como gráficos 3D e suporte a banco de dados nativo, também estão disponíveis em outras plataformas móveis (ANDROID, 2020) (MARWEDEL, 2021).

Segundo dados obtidos a partir do site oficial da plataforma Android (2020) as camadas da arquitetura de seu sistema estão dispostos na Figura 6 e podem ser descritas da seguinte maneira:

Aplicativos: A camada de aplicativos é o topo da pirâmide da arquitetura do sistema operacional Android, composto pelos aplicativos nativos do sistema. Cliente de e-mail, despertador, calendário, jogos, mapas, browser e internet, etc são exemplos disso. Os aplicativos do sistema funcionam como aplicativos para usuários e fornecem recursos básicos que os desenvolvedores podem acessar por meio de seus próprios aplicativos.

Framework: Um *framework* nativo fornece aos desenvolvedores as mesmas *Applications Programming Interface* (APIs) – Interface de Programação de Aplicativos – que são usadas para criar aplicativos Android nativos. Esse *framework* permite que o programador tenha o

Figura 6 – Arquitetura do sistema operacional Android



Fonte: Android (2020).

mesmo acesso ao sistema que os aplicativos no cadastro de aplicativos. Essas APIs são os blocos de programação necessários para criar aplicativos Android, simplificando a reutilização de componentes e serviços do sistema.

Bibliotecas e serviços: Essas bibliotecas são responsáveis por fornecer funções para manipulação de áudio, vídeo, gráficos, bancos de dados e navegadores. O OpenGL ES pode ser acessado por meio da API Java OpenGL do Android para adicionar a capacidade de desenhar e manipular gráficos 2D e 3D ao seu aplicativo. Existem outros serviços usados em camadas de nível superior, como uma máquina virtual Java Dalvik. A maioria dessas bibliotecas e serviços são escritos em C e C++.

Android Runtime (ART): Foi projetado para executar uma variedade de máquinas virtuais em dispositivos com pouca memória executando arquivos Dalvik Executable (DEX), um formato de bytecode projetado especificamente para Android e otimizado para o uso de pouca memória. Alguns principais recursos do ART:

- Compilação "*Ahead-Of-Time*" (AOT) e "*Just-In-Time*" (JIT)

- Coleta de lixo (GC) otimizada
- Android 9 (nível de API 28) ou superior, a conversão dos arquivos com o formato DEX de um pacote de aplicativos utiliza um código de máquina mais compacto.
- Melhor compatibilidade de depuração, incluindo um criador de perfil, exceções de diagnóstico detalhadas e geração de relatórios de erros, bem como a capacidade de definir pontos de controle para monitorar determinados campos.

O Android também inclui uma coleção das bibliotecas de tempo de execução mais importantes, que fornecem a maioria das funcionalidades da linguagem de programação Java, bem como alguns recursos de linguagem Java 8 usados pela biblioteca Java da API.

Camada de abstração de hardware (HAL): Camada que fornece interfaces padrão que expõem os recursos de hardware do dispositivo para a estrutura de API Java de nível superior. Uma HAL é composta de módulos de bibliotecas que implementam uma interface para um tipo específico de componente de hardware, como um módulo de câmera ou Bluetooth. Quando uma API do framework faz uma chamada para acessar o hardware do dispositivo, o sistema Android carrega o módulo da biblioteca para aquele componente de hardware.

Kernel Linux: Essa camada também é responsável pela separação entre o hardware e os aplicativos, bem como os serviços centrais do Android, como gerenciamento de memória e gerenciamento de processos. Muitas funções do kernel são usadas diretamente pelo Android. Por exemplo, o (ART) depende do kernel Linux para realizar tarefas como codificação e gerenciamento de memória de baixo nível. Muitas mudanças foram feitas para otimizar a memória e o tempo de processamento. Essas mudanças incluem novos dispositivos de driver, adições ao sistema de gerenciamento de energia e um sistema que permite que os processos sejam concluídos de maneira baseada em critérios quando a memória é limitada.

2.5 FLUTTER

O Flutter é um SDK móvel de código aberto desenvolvido pela Google, com foco principal em permitir que todos criem aplicativos móveis mais sofisticados. A ferramenta simplifica o desenvolvimento e planejamento de aplicativos móveis, independentemente da experiência do desenvolvedor em programação *web* ou desenvolvimento nativo (WINDMILL, 2020).

A linguagem de programação utilizada no Flutter é o Dart, criada pelo Google em 2011. O Dart possui recursos como suporte a diversas funcionalidades de programação, interfaces,

classes abstratas e coleções. Sua sintaxe é baseada na linguagem C e é otimizada para o desenvolvimento de interfaces de usuário (WINDMILL, 2020).

O Flutter permite desenvolver aplicativos para diferentes sistemas operacionais móveis, como Android, iOS e ChromeOS, utilizando o mesmo código. É uma plataforma abrangente que oferece recursos como mecanismo de renderização, componentes de interface do usuário, estruturas de teste, ferramentas e roteadores para a criação de aplicativos (WINDMILL, 2020).

O *JavaScript bridge*, utilizada na maioria das opções multiplataforma, constitui um gargalo significativo no desenvolvimento e desempenho de aplicações. A rolagem não é fluida, os aplicativos nem sempre apresentam um desempenho adequado e são de difícil depuração. Por outro lado, o Flutter é compilado para código nativo real e é renderizado utilizando o mesmo mecanismo utilizado pelo Chrome para renderização (chamado Skia), eliminando assim a necessidade de traduzir o Dart em tempo de execução. Essa abordagem garante que os aplicativos não percam desempenho ou produtividade ao serem executados em dispositivos dos usuários (WINDMILL, 2020).

Ao contrário do desenvolvimento nativo, onde o compartilhamento de código é limitado, o Flutter permite que aplicativos web e móveis compartilhem todo o código, com exceção das visualizações específicas de cada cliente. Essa capacidade de compartilhamento é possível por meio do uso de injeção de dependência, permitindo a execução de um aplicativo *AngularDart* e um aplicativo Flutter com os mesmos modelos e controladores (WINDMILL, 2020).

Em termos práticos, o compartilhamento de código proporciona uma alta produtividade no desenvolvimento de aplicativos. Ao desenvolver recursos móveis primeiro e compartilhar a lógica de negócio entre web e móvel, uma vez que o recurso móvel é implementado, é necessário apenas escrever as visualizações específicas para a web que esperam os mesmos dados do controlador. Essa abordagem economiza tempo e esforço, permitindo que os desenvolvedores se concentrem na criação de experiências de usuário consistentes em diferentes plataformas (WINDMILL, 2020).

Em suma, o compartilhamento de código no Flutter é uma vantagem que aumenta a eficiência do desenvolvimento de aplicativos, promovendo a reutilização de lógica de negócio e acelerando o processo de implementação de recursos em diferentes plataformas (WINDMILL, 2020).

2.6 APRENDIZADO DE MÁQUINA

O aprendizado de máquina (AM) é definido por Mitchell e Mitchell (1997) como: “A capacidade de melhorar o desempenho na realização de alguma tarefa por meio da experiência”. Para Alpaydin (2020) é o processo de ensinar aos computadores como otimizar um critério de desempenho com base em experiências anteriores ou dados de exemplo. A partir de um modelo estabelecido com alguns parâmetros, o aprendizado é o processo de execução de um programa de computador para otimizar os parâmetros do modelo usando dados de treinamento ou experiência anterior.

A principal tarefa do aprendizado de máquina é inferir a partir de uma amostra, com isso é empregado a teoria estatística na construção de modelos matemáticos. A ciência da computação serve a um propósito duplo: é necessário de algoritmos eficientes no treinamento para resolver o problema de otimização, bem como armazenar e manipular uma grande quantidade de dados. Em segundo lugar, após a parte de aprendizado do modelo matemático, o modelo deve ser representado e inferido usando uma abordagem eficiente (ALPAYDIN, 2020).

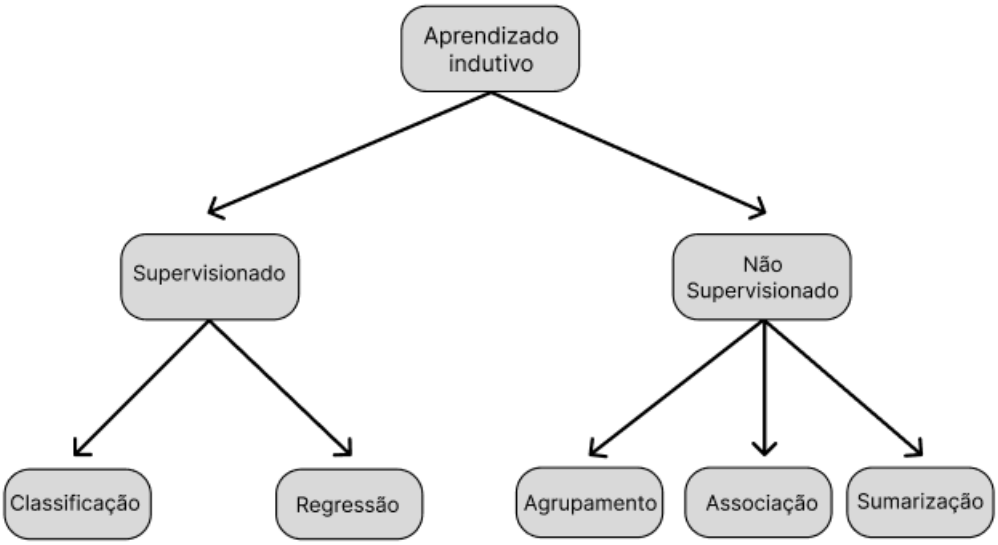
Os algoritmos de aprendizado de máquinas podem ser preditivos para fazer previsões futuras, descritivos para aprender com os dados ou ambas (ALPAYDIN, 2020). Segundo Faceli *et al.* (2011) nas tarefas de classificação, o objetivo é encontrar uma função (conhecida também como modelo ou hipótese) a partir de dados de treinamentos que possam ser usados para prever um rótulo ou valor que caracterize um novo exemplo, com base nos valores de seus atributos de entrada e saída.

Os algoritmos ou métodos de AM usados a partir desta abordagem descrita anteriormente indicam modelos preditivos. Esses algoritmos aderem ao paradigma de aprendizado supervisionado, o qual refere-se à simulação na presença de um “supervisor externo” que está familiarizado com a saída (rótulo) desejado para cada exemplo. Como resultado, um supervisor externo pode avaliar a capacidade da hipótese indutiva de prever o valor de saída para novos exemplos (FACELI *et al.*, 2011).

O objetivo das tarefas de descrição é explorar ou descrever um conjunto de dados. Os algoritmos AM usados nessas tarefas não fazem uso do atributo saída. Como resultado, eles continuam seguindo o paradigma de aprendizado não supervisionado. Por exemplo, uma tarefa de descrição de agrupamento de dados tem como objetivo a descoberta de grupos semelhantes de objetos em um conjunto de dados. Outra tarefa é encontrar regras associativas que incluam um conjunto de atributos a outro conjunto de atributos (FACELI *et al.*, 2011).

A Figura 7 descreve uma hierarquia de aprendizado com base nos vários tipos de tarefas. O aprendizado indutivo, processo pelo qual são feitas as generalizações com base em dados, aparece no topo. A seguir estão os tipos de aprendizagem supervisionada e não supervisionada. As tarefas supervisionadas distinguem-se pelo tipo de rótulos de dados utilizados: discretos no caso de classificação os quais utilizam dados rotulados com o objetivo de gerar regras ou classificadores, e contínuos no caso de regressão, ao invés gerar classificadores, a regressão é utilizada para prever um valor contínuo. As tarefas descritivas dividem-se em três categorias: agrupamento, em que os dados são agrupados de acordo com a sua semelhança; sumarização, em que o objetivo é encontrar uma descrição concisa e simples para um conjunto de dados; e associação, em que o objetivo é encontrar padrões comuns de associação entre os atributos de um conjunto de dados (FACELI *et al.*, 2011).

Figura 7 – Hierarquia de Aprendizado



Fonte: Fonte adaptada Faceli *et al.* (2011).

O elemento de aprendizagem que não é abordado anteriormente é o aprendizado por reforço. O objetivo deste elemento é reforçar ou compensar uma ação positiva enquanto pune uma ação negativa (FACELI *et al.*, 2011).

2.6.1 Tipos de Classificadores

Existem diversos tipos de classificadores utilizados no Aprendizado de Máquina, dentre eles é possível citar *eager* e *lazy algorithms*. *Eager learning* é um método de aprendizado no qual o sistema se esforça para desenvolver classificadores independentemente da entrada durante o

treinamento. Entretanto, *lazy algorithms* não calculam um modelo quando recebem um conjunto de treinamento; ao invés disso, eles esperam até receber uma instância de teste para efetuar a classificação (ALPAYDIN, 2020).

2.6.2 Algoritmo *k-Nearest Neighbors* (k-NN)

Dentre os mais populares algoritmos de classificação destaca-se o k-NN, ele é considerado um algoritmo *lazy* pois não aprende um modelo compacto para os dados, ao invés disso, ele acessa as instâncias da base de dados. Uma das vantagens deste algoritmo é que ele pode ser usado para resolver tanto problemas de classificação, quanto de regressão direta, sem a necessidade de mudanças significativas (FACELI *et al.*, 2011)

O objetivo deste algoritmo é considerar os k objetos do conjunto de treinamento que estão mais próximos do ponto de teste x_t(objeto em que será realizado a classificação), em que k é um parâmetro do algoritmo. Quando o valor de k é superior a 1, k vizinhos são obtidos para cada ponto de teste. Cada vizinho vota em uma das classes, as previsões dos diferentes vizinhos são combinadas para classificar o ponto de teste (FACELI *et al.*, 2011).

Em problemas de classificação, quando a classe atribui valores a um grupo discreto, cada observador vota em uma das classes. O objeto de teste é atribuído à classe mais popular. Esta abordagem é equivalente a $\hat{f}(x_t) \leftarrow moda(f(x_1), f(x_2) \cdots, f(x_k)$, o que se justifica porque a constante que minimiza a função de custo $0-1$, é a $moda$ (FACELI *et al.*, 2011)..

Em que a função $\hat{f}(x_t)$ é a hipótese aplicada ao ponto de teste x_t, seguido de $f(x_k)$ que é a função objetivo aplicada ao objeto x_t.

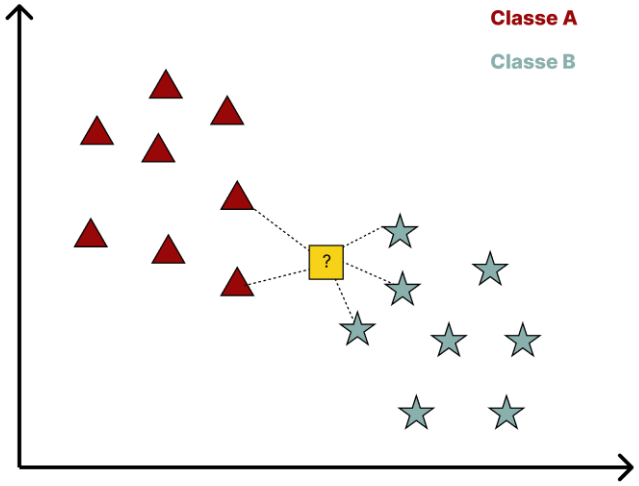
E a moda ponderada é representada por: $y_t = arg\max_{c \in Y} \sum_{i=1}^{k} w_i I(c,y_i)$, com $w_i = \frac{1}{d(x_t,x_i)}$ e $I(a,b)$, é uma função que retorna 1 se e somente se $a = b$.

Em que y_i é a classe do exemplo x_i, w_i representa o peso associado ao exemplo x_i, c é a classe com maior moda ponderada e $d(x_t,x_i)$ é a distância euclidiana:

$$d(x_t, x_i) = \sqrt{\sum_{l=1}^{d} (x_t^l - x_j^l)^2}$$

A Figura 8 representa de uma forma ilustrativa o funcionamento do algoritmo, neste exemplo ocorre uma verificação entre a instância não rotulada representada por "?" em relação às cinco instâncias mais próximas, as quais já possuem um rótulo, como a classe ilustrada como uma estrela aparece com maior frequência, a instância não rotulada vai fazer parte dessa classe.

Figura 8 – Exemplo do algoritmo K-NN em um plano 2D



Fonte: Autoria Própria.

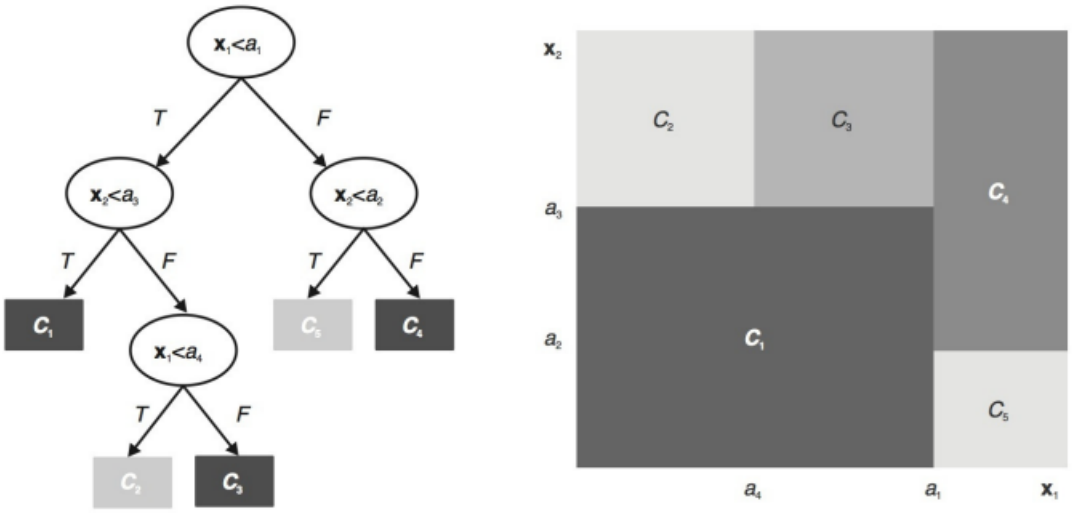
2.6.3 Árvore de Decisão

Uma árvore de decisão é um modelo hierárquico para aprendizado supervisionado no qual a região local é identificada em um número menor de etapas por meio de uma série de divisões recursivas, com isso um problema complexo é dividido em problemas mais simples (FACELI *et al.*, 2011). Segundo Alpaydin (2020), os nós de decisão internos e as folhas terminais formam uma árvore de decisão. Cada nó de decisão m implementa uma função de teste $f_m(x)$ que rotula os ramos com resultados discretos.

A Figura 9 caracteriza uma árvore de decisão, sua divisão é definida pelos atributos x_1 e x_2. Dada uma entrada, descrita na figura por valores que são representados por, a_1,a_2, a_3 e a_4, um teste é executado em cada nó e uma das ramificações é classificada com base no resultado. O procedimento de classificação começa na raiz da árvore e continua até que um nó folha seja alcançado. Os nós folha são os nós finais da árvore e contêm os rótulos de classificação, representados por C_1, C_2, C_3 e C_4 que se tornam a saída final do processo de classificação.

Após a realização das divisões nos nós da árvore, é possível observar os rótulos demarcados no espaço de objetos. Isso significa que, ao realizar as divisões com base nos atributos x1 e x2, a árvore de decisão é capaz de determinar regiões no espaço de objetos e atribuir rótulos específicos a cada região. Esses rótulos demarcados no espaço de objetos indicam as decisões de classificação tomadas pela árvore.

Figura 9 – Árvore de decisão e os rótulos demarcados no espaço de objetos



Fonte: Faceli *et al.* (2011).

2.6.3.1 Regras de divisão para classificação

Segundo Alpaydin (2020) uma medida de impureza é utilizada para quantificar a qualidade de uma divisão em uma árvore de decisão. Uma divisão é pura se, após a divisão, todas as instâncias que selecionam uma ramificação pertencem à mesma classe para todas as ramificações . N_m é o número de instâncias de treinamento que atingiram o nó m. E para o nó raiz é o número de treinamentos que atingiram o nó m é N. N_m^i derivado de N_m pertence a classe C_i, em que i é referente a quantidade de instâncias de treinamento que atingiram o nó m então $\sum_i N_m^i = N_m$. Dado que uma instância que atinge o nó m a probabilidade estimada que essa instância pertença a classe C_i é dada por.

$$\hat{P}(C_i|x,m) \equiv p_m^i = \frac{N_m^i}{N_m} \tag{1}$$

Se p_m^i (probabilidade estimada que essa instância pertença a classe C_i) para todo i for 0 ou 1, o nó m é puro. É 0 se nenhuma das instâncias que atingem o nó m for da classe Ci, e é 1 se todas forem. Se a divisão for pura, não há mais a necessidade de dividir, então pode-se adicionar um nó folha rotulado com a classe para a qual p_m^i é 1. Uma opção é a entropia (função usada para medir a impureza). A fórmula da entropia pode ser descrita pela Equação 2:

$$I_m = -\sum_{i=1}^{K} p_m^i \log_2 p_m^i \tag{2}$$

em que 0 log 0 é igual a 0 e K é o número de classes. Na teoria da informação, a entropia refere-se ao menor número de bits necessários para codificar o código de classe de uma instância. Em um

problema com duas classes, se $p^1 = 1$ e $p^2 = 0$, é possível concluir que todos os exemplos são da classe C_1, com isso a entropia é 0. Se $p^1 = p^2 = 0{,}5$ é necessário enviar um bit para sinalizar um dos dois cenários, nesse caso a entropia é 1 (ALPAYDIN, 2020).

É possível criar códigos que consomem menos de um bit por mensagem entre esses dois extremos, obtendo códigos mais curtos para classe mais provável e códigos mais longos para os menos prováveis. Quando a classe for $K > 2$, é possível utilizar a mesma explicação descrita anteriormente, com a maior entropia sendo $\log_2 K$ no momento em que $p_m^i = \frac{1}{K}$. Contudo, a entropia não é a única métrica disponível. Para um problema com duas classes, se $p^1 \equiv p$ e $p^2 = 1 - p$, $\phi(p,1-p)$ é uma função não negativa que avalia a impureza de uma divisão se satisfazer as seguintes propriedades (DEVROYE; GYÖRFI; LUGOSI, 1996):

$\phi(1/2, 1/2) \geq \phi(p, 1-p)$, para todo $p \in [0,1]$. $\phi(0,1) = \phi(1,0) = 0$. $\phi(p, 1-p)$ é crescente quando p $[0,1/2]$ e decrescente quando p for $[1/2,1]$.

Sendo p (probabilidade estimada que essa instância pertença a uma determinada classe).

Com isso é possível obter as seguintes representações para impureza.

1. Entropia

$$\phi(p, 1-p) = -p \log_2 p - (1-p) \log_2 (1-p) \tag{3}$$

2. Gini

$$\phi(p, 1-p) = 2p(1-p) \tag{4}$$

3. Erro de classificação

$$\phi(p, 1-p) = 1 - \max(p, 1-p) \tag{5}$$

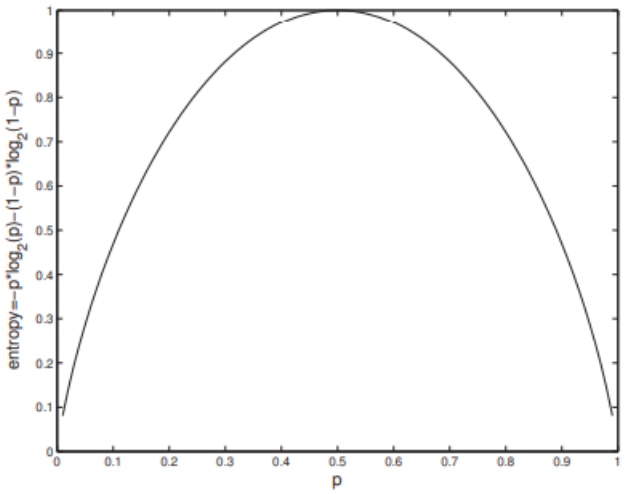
Segundo Devroye, Györfi e Lugosi (1996), não há diferença estatisticamente significativa entre esses três indicadores.

A Figura 10 representa a variação da entropia para um problema com duas classes, em relação ao p (probabilidade estimada que essa instância pertença a uma determinada classe).

Segundo Alpaydin (2020) se o nó m não for puro, as instâncias devem ser separadas para reduzir a impureza, e é possível dividir em vários atributos diferentes. Existem várias posições diferentes de divisão para uma característica numérica. O objetivo é construir a menor árvore, então é necessário buscar a divisão que reduza a sua impureza. Se após a divisão, os subgrupos estiverem mais próximos do puro, menos divisões serão necessárias.

Um exemplo descrito por Alpaydin (2020) supõe que o nó m, N_m utilize o ramo j, então é possível obter N_{mj}. Para um atributo discreto com n valores, existem n resultados, e para um atributo numérico, existem dois resultados ($n = 2$), em ambos os casos satisfazendo essa relação

Figura 10 – Gráfico da função de Entropia para um problema com duas classes



Fonte: Alpaydin (2020).

é obtida: $\sum_{i=1}^{K} N_{m_j}^i = N_m^i$.

N_{mj} derivado de $N_{m_j}^i$ pertence a classe C_i : $\sum_{i=1}^{K} N_{m_j}^i = N_{mj}$. De forma similar é obtido que $\sum_{j=1}^{n} N_{m_j}^i = N_m^i$.

Com isso, dado que no nó m o teste retorna o valor do ramo j, a probabilidade que o valor seja da classe C_i é dada por:

$$\hat{P}(C_i|x,m,j) \equiv p_{m_j}^i = \frac{N_{m_j}^i}{N_{m_j}} \tag{6}$$

A impureza total após a divisão é:

$$I'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p_{m_j}^i \log_2 p_{m_j}^i \tag{7}$$

O algoritmo que representa um pseudocódigo da árvore de decisão para classificação é exibido a seguir.

O Algoritmo 1 representa o pseudocódigo para a árvore de decisão, modelo de classificação. Neste exemplo de (ALPAYDIN, 2020) é possível observar que a cada etapa de construção da árvore, é escolhido a divisão que causa a maior diminuição da impureza representado pela Equação 2. A entropia total dos dados é calculada utilizando a Equação 7. Com isso a construção da árvore continua recursivamente e em paralelo para todos os ramos que não são puros, até que todos se tornem puros.

Um problema dessa abordagem é que, como ela tem como objetivo obter a máxima diminuição da impureza, há muitas ramificações na árvore de decisão final. Nós com diversas ramificações são difíceis de compreender, e também vão contra o objetivo de dividir discriminantes

Algoritmo 1 – Pseudocódigo, árvore de decisão para classificação

```
Garantir: gerarArvore(X)
se entropiaDoNo(X) < θ1 /* Equação 2 então
    Criar folha rotulada pela classe majoritária em X
    Return
finaliza se
i ← dividirAtributo(X)
para Cada ramo de x_i faz
    Encontre o nó folha X_i
    gerarArvore(X_i)
finaliza para
Garantir: dividirAtributo(X)
MinEnt ← MAX
para Todos os atributos i = 1, ··· ,d faz
    se x_i é discreto com n valores então
        Divida X em X_1, ··· ,X_n por x_i
        e ← dividirEntropia(X_1, ··· ,X_n)  /*Equação 7*/
        if e < MinEnt MinEnt ← e; bestf ← i
    senão {/* x_i é numérico */}
        Para todas as possibilidades de divisão
        Dividir X em X_1, X_2 em x_i
        e ← dividirEntropia(X_1, X_2
        if e < MinEnt MinEnt ← e; bestf ← i
    finaliza se
finaliza para
Return bestf
```

Fonte: Fonte adaptada Alpaydin (2020).

de classe em julgamentos fáceis, e podem causar overfitting (quando o sistema decora os dados de treinamento, possuindo um resultado de precisão excelente e para o modelo de teste possui um desempenho ruim).

Para evitar o *overfitting*, a construção da árvore para quando os nós são puros o suficiente, ou seja, um subconjunto de dados não é mais duvidoso se $I < \theta_I$. Isso significa que não é necessário que $p^i_{m_j}$ seja exatamente 0 ou 1, mas próximo o suficiente, com um limiar p. Visto isso, um nó folha é construído e a classe com o maior $p^i_{m_j}$ é marcada (ALPAYDIN, 2020). Em que I é a representação para a entropia calculada e θ_I é o parâmentro de complexidade, dessa maneira a árvore não se torna complexa a ponto de ocorrer um overfitting.

Em outras palavras, para evitar esse problema, a construção da árvore de decisão é interrompida quando os nós são puros. Isso significa que um subconjunto de dados não é mais considerado duvidoso se a medida de impureza for menor que um limiar pré-definido. A medida de impureza é calculada para determinar o quão misturados estão os dados em um nó. Se a impureza for menor que o limiar estabelecido, um nó folha é criado e é atribuída a ele a classe mais comum entre os exemplos desse nó. Isso ajuda a evitar que a árvore se torne excessivamente complexa e se ajuste excessivamente aos dados de treinamento.

3 MATERIAIS E MÉTODO

Neste capítulo são apresentados os materiais utilizados para desenvolver a pesquisa e a metodologia utilizada para abordar a questão levantada no presente trabalho. Na seção materiais, as principais ferramentas para desenvolver o aplicativo são descritas. Na seção método, as técnicas focadas em aprendizagem de máquinas, assim como o desenvolvimento do aplicativo para a captura da base de dados serão descritas.

3.1 MATERIAIS

No Quadro 1, são listados os principais softwares utilizados para o desenvolvimento do presente trabalho.

Quadro 1 – *Softwares* a serem utilizados no desenvolvimento do trabalho

Ferramenta/tecnologia	Versão	Finalidade
Windows	10	Sistema Operacional
Flutter	2.10.5	Framework para desenvolvimento de aplicativos mobile
Dart	2.16.2	Linguagem de programação
Path Provider	9.0.2	Biblioteca para o armazenamento de dados em arquivos externos e internos
Geolocator	9.0.2	API utilizada para receber a real localização do usuário
Sensors plus	2.0.2	API utilizada para coletar os dados do acelerômetro, giroscópio e magnetômetro
Python	3.10.2	Linguagem de programação
Scikit-learn	1.1.1	Biblioteca de aprendizado de máquina
Pandas	1.4.3	Biblioteca para manipulação de dados
Google Colab	2023	Implementação dos códigos em Python
Weka	3.9.6	Ferramenta de aprendizado de máquina

Fonte: Autoria própria.

Durante o desenvolvimento do aplicativo, foi escolhido o *framework Flutter* devido à sua ampla disponibilidade de bibliotecas e, principalmente, sua compatibilidade de código com os dois principais sistemas operacionais móveis, Android e iOS.

Durante a coleta de dados, os valores de velocidade, giroscópio, magnetômetro e posição (longitude e latitude) foram registrados em um arquivo de texto (TXT) no dispositivo móvel utilizado para a coleta. Após o armazenamento dos dados, o arquivo TXT foi utilizado como entrada para as etapas de pré-processamento e treinamento dos algoritmos de aprendizagem de máquina.

Para realizar o treinamento e classificação da base de dados coletada pelo aplicativo,

foram escolhidas as seguintes ferramentas: *WEKA* (*Waikato Environment for Knowledge Analysis*), uma ferramenta desenvolvida em Java, e o Google Colab, cuja implementação dos algoritmos de classificação foi realizada em *Python*.

Essas escolhas foram feitas levando em consideração a facilidade de uso, a disponibilidade de recursos e a compatibilidade com as etapas de pré-processamento e treinamento dos dados. O *WEKA* é amplamente utilizado para análise de dados e possui uma variedade de algoritmos de aprendizagem de máquina implementados. O Google Colab é uma plataforma baseada em nuvem que permite executar os projetos em Python, o que facilita a implementação e execução dos algoritmos de classificação.

Quadro 2 – *Hardwares* a serem utilizados no desenvolvimento do trabalho

Ferramenta/tecnologia	Finalidade
Acer Nitro 5	Notebook utilizado para o desenvolvimento do aplicativo
GPS	Sensor recebe a posição atual do usuário
Acelerômetro	Instrumento capaz de medir a aceleração do usuário
Giroscópio	Sensor que detecta a velocidade angular do aparelho
Magnetômetro	Sensor utilizado para medir a intensidade, direção e sentido dos campos magnéticos
Xiaomi Poco X3 GT	Dispositivo móvel contendo todos os sensores e instrumentos de navegação citados nesta tabela. Utilizado para realizar os testes e coleta de dados

Fonte: Autoria própria.

Para a coleta dos dados de geolocalização, magnetômetro e giroscópio, o aparelho citado no Quadro 2, Xiaomi Poco X3 GT foi utilizado, é importante ressaltar que variações na precisão dos dados coletados podem ocorrer em decorrência dos sensores utilizados no aparelho.

3.1.1 Base de Dados

Um dos desafios desse presente trabalho é atuar na identificação de fatores que possam diferenciar as ações referentes ao ato de estacionar um veículo ou realizar qualquer outra ação diária.

Uma das contribuições do trabalho desenvolvido consiste na obtenção da base de dados, a qual possui uma grande influência no resultado do projeto apresentado. Uma das principais preocupações no momento da coleta era para que a alocação das colunas de cada variável fosse realizada de maneira correta, e também, para que os valores fossem separados por vírgulas. A base de dados utilizada nos treinamentos foi montada com as informações que variam com as seguintes características:

- Velocidade.
- Giroscópio, obtendo os três principais eixos de orientação, X, Y e Z.
- Magnetômetro, obtendo os três principais eixos de orientação, X, Y e Z.
- Posição atual contendo os valores da latitude e longitude.

Composição da base de dados: A composição da base de dados utilizada neste estudo consiste em dados coletados pelo dispositivo móvel descrito no Quadro 2. A base de dados é composta por informações de velocidade, giroscópio, magnetômetro, latitude e longitude. Durante a etapa de treinamento, foi identificado que a inclusão dos dados de latitude e longitude levava os algoritmos de treinamento a superajustar os dados, resultando em *overfitting* na classificação.

Ao realizar o treinamento inicial das bases de dados no WEKA utilizando o algoritmo *Random Forest*, observou-se que, ao utilizar a base de dados 1, representada pela Tabela 1, que antes da exclusão dos dados de latitude e longitude, o algoritmo obteve uma acurácia de 99,53%. Isso indica que o modelo foi capaz de se ajustar quase perfeitamente aos dados de treinamento, o que pode ser um indicativo de *overfitting*. O mesmo ocorreu com o algoritmo k-NN, que obteve uma acurácia de 98,70%. Portanto, os dados de latitude e longitude não estão presentes nas Tabelas 1 e 2 e não foram utilizados nas etapas subsequentes do trabalho.

Tabela 1 – *B*ase de Dados 1

	Velocidade	Giroscopio (X)	Giroscopio (Y)	Giroscopio (Z)	Magnetometro (X)	Magnetometro (Y)	Magnetometro (Z)	Estado
0	22.915005	0.609675	0.529100	0.833387	24.300001	5.700000	52.500004	1
1	21.184531	-1.570938	-2.929850	1.930637	24.656252	-2.587500	31.256250	1
2	21.184531	-1.679562	-0.344438	-0.522637	54.112503	54.112503	5.756250	1
...	...	...	...	...	...	...	...	...
11191	22.440030	0.009212	0.052387	0.004262	23.812500	9.956250	3.843750	0
11192	22.440030	0.003850	-0.012650	0.020350	23.643751	9.581250	4.012500	0
11193	22.440030	0.022000	-0.008388	-0.015950	23.700001	9.543751	4.143750	0

Fonte: Autoria própria.

Tabela 2 – Base de Dados 2

	velocidade_1	giroscopio_x_1	giroscopio_y_1	giroscopio _z_1	magnetometro_x_1	magnetometro_y_1	magnetometro_z_1	velocidade_2	giroscopio_x_2	...	velocidade_119	giroscopio_x_119
0	22.915	0.609	0.529	0.833	24.300	5.700	52.500	21.184	-1.570	...	2.61e-08	-0.096
1	1.68e+01	-0.094	0.223	-0.051	-2.812	28.387	33.656	16.896	0.189	...	5.53e-15	0.014
2	2.95e-18	0.068	-0.115	0.118	-13.387	29.156	35.756	12.992	0.132	...	1.09e-10	0.014
...	...	...	...	...	...	...	...	...	...	...	...	...
89	4.35e+01	0.009	0.042	0.004	-0.281	-5.325	31.425	43.501	0.016	...	4.160	0.007
90	4.160	0.002	0.027	-0.046	-2.700	-8.381	31.143	4.160	-0.003	...	4.86e+01	-0.009
91	4.03e+01	-0.026	-0.009	0.028	-0.787	0.581	44.587	40.304	0.040	...	2.40e+01	0.113

Fonte: Autoria própria.

Apesar do treinamento contendo os dados de latitude e longitude resultar em uma elevada taxa de classificação assertiva para esses algoritmos, essa alta acurácia pode não se

traduzir em uma boa capacidade de generalização para novos dados, especialmente em situações em que a localização geográfica difere daquela presente nos dados de treinamento. Isso pode representar um problema, uma vez que o objetivo é desenvolver um sistema capaz de distinguir atividades de estacionamento em diferentes contextos e locais.

Coleta dos dados: A coleta de dados foi dividida em duas etapas distintas. Na primeira etapa, os dados foram coletados durante a condução do veículo, registrando todas as ações desde a aceleração até a parada total, simulando uma situação de estacionamento, totalizando **45** coletas. Na segunda etapa, a coleta foi realizada em duas partes: na primeira parte, foram realizadas atividades básicas do dia a dia, como caminhar e correr. Já na segunda parte, os dados foram coletados enquanto um veículo era dirigido, registrando ações como paradas em cruzamentos, variações de velocidade e outras situações corriqueiras do trânsito, totalizando **47** coletas. Ambas as etapas foram realizadas em ambientes distintos da cidade, simulando diversas situações reais. Cada uma das 92 coletas foi registrada ao longo de 60 segundos, com uma frequência de coleta de 0,5 segundos, obtendo assim 119 dados de velocidade, giroscópio (eixos x, y, z), magnetômetro (eixos x, y, z) e latitude/longitude em cada coleta. É importante destacar que, devido às limitações do sensor, o último meio segundo não foi considerado na coleta de dados.

A Tabela 1 representa um fragmento da base de dados, contendo todos os dados das coletas. Cada linha nessa tabela é classificada de forma única, representando uma instância distinta de atividade de estacionamento. Por outro lado, a Tabela 2 representa uma forma diferente de organização da base de dados, em que cada coleta de dados individual é representada por uma única linha. Isso significa que as 119 linhas de cada coleta na Tabela 1 são transformadas em colunas na Tabela 2.

Essa diferenciação na estrutura das bases de dados permite explorar diferentes abordagens de análise e modelagem, adaptando-as de acordo com os objetivos e necessidades do estudo. Ambas as representações podem ser úteis para entender os padrões e relações presentes nos dados de atividades de estacionamento.

Análise descritiva dos dados: A análise descritiva dos dados tem como objetivo fornecer uma visão geral das características presentes na base de dados utilizada neste estudo. Nessa análise, foram calculadas as médias, valor máximo, valor mínimo e desvio padrão de cada coluna da base de dados 1. Vale ressaltar que essa análise não foi realizada na base de dados 2 pelo motivo de possuir 834 colunas além dos dados serem os mesmos da base 1, dispostos de maneira distinta.

Os resultados obtidos na análise descritiva da coluna "Velocidade" indicaram um valor

mínimo de 0, valor máximo de 87, média de 16.89 e desvio padrão de 15.95. Quanto à coluna "Giroscópio (X)", os atributos encontrados foram valor mínimo de -6.706, valor máximo de 6.08, média de 0.006 e desvio padrão de 0.633. Na coluna "Giroscópio (Y)", os atributos foram valor mínimo de -10.729, valor máximo de 7.544, média de -0.007 e desvio padrão de 0.735. Já na coluna "Giroscópio (Z)", os atributos foram valor mínimo de -5.82, valor máximo de 5.901, média de 0.009 e desvio padrão de 0.644.

No que se refere à coluna "Magnetômetro (X)", foram encontrados os seguintes atributos: valor mínimo de -10.729, valor máximo de 7.544, média de 10.857 e desvio padrão de 19.176. Na coluna "Magnetômetro (Y)", os atributos foram valor mínimo de -68.513, valor máximo de 108.056, média de 11.486 e desvio padrão de 18.58. Por fim, na coluna "Magnetômetro (Z)", os atributos encontrados foram valor mínimo de -78.001, valor máximo de 87.356, média de 3.973 e desvio padrão de 22.012. Esses resultados fornecem informações importantes sobre as características e variações presentes nos dados utilizados neste estudo.

3.2 MÉTODO

O método do presente trabalho visa a possibilidade de distinção, por meio de algoritmos de aprendizagem de máquina, das atividades de estacionar um veículo ou estar realizando qualquer outra atividade distinta, seja ela dirigindo um veículo ou não.

3.2.1 Implementação do aplicativo

O aplicativo foi desenvolvido com o objetivo principal de coletar dados de velocidade, giroscópio, magnetômetro, latitude e longitude, utilizando as bibliotecas Geolocator e Sensors Plus, conforme descrito na Tabela 1. Nesta seção, serão apresentados em detalhes os métodos utilizados para capturar essas informações, que foram usadas nas etapas de criação da base de dados e treinamento dos algoritmos de classificação.

Durante o desenvolvimento do aplicativo, foi necessário implementar uma função para obter os dados de localização (latitude e longitude) por meio da biblioteca Geolocator, assim como os dados dos sensores de velocidade, giroscópio e magnetômetro, que são obtidos usando a biblioteca Sensors Plus. O Código-fonte 1 descreve em detalhes como foi realizada a coleta desses dados.

No Código-fonte 1, a biblioteca Geolocator é utilizada para obter os dados da latitude e

Código-fonte 1 – Listener dos sensores

```
1  _adicionarListenerLocalizacao() {
2
3     Geolocator.getPositionStream(
4             locationSettings: LocationSettings(accuracy:
    LocationAccuracy.best))
5         .listen((Position position) {
6       setState(() {
7         posicaoAtual = position;
8         _posicaoCamera = CameraPosition(
9             target: LatLng(position.latitude, position.longitude),
    zoom: 18);
10
11        _velocidadeAtual = position.speed ?? 0.0;
12        _velocidadeAtual = _velocidadeAtual * 3.6;
13
14
15        gyroscopeEvents.listen((GyroscopeEvent event) {
16          _giroscopio = event;
17        });
18
19
20        magnetometerEvents.listen((MagnetometerEvent event) {
21          _magnetometro = event;
22        });
23
24        _movimentarCamera();
25      });
26    });
27  }
```

Fonte: Autoria própria.

longitude atual do usuário com maior precisão. Na linha 4, é configurado o nível de acurácia da localização para "melhor" (*accuracy: LocationAccuracy.best*) por meio do *LocationSettings*. A posição atual é recuperada na linha 9 e a atualização da posição é mostrada na tela. Nas linhas 11 e 12, o sensor de velocidade é utilizado, e na linha 12, a velocidade é convertida de metros por segundo para quilômetros por hora. Entre as linhas 15 e 22, a biblioteca *sensors_plus* é empregada para obter os dados do giroscópio e do magnetômetro. Por fim, na linha 24, é chamada a função responsável por movimentar a câmera com base na atualização do usuário.

Quando o botão de iniciar a coleta for selecionado, acionará a função descrita no Código-fonte 2.

Na função _coletarDados() descrita pelo Código-fonte 2, primeiramente, na linha 4, a variável global responsável por exibir mensagens na tela para o usuário é atualizada com a informação de que a coleta de dados foi iniciada. Entre as linhas 6 e 9, é realizado o procedimento

Código-fonte 2 – Iniciar a coleta

```
1
2   void _coletarDados() async {
3     setState(() {
4       mensagemTela = "Coleta Iniciada";
5     });
6     Directory? directory = await getExternalStorageDirectory();
7     if (directory != null) {
8       filePath = directory.path + '/$fileName';
9     }
10
11    _timer = Timer.periodic(Duration(milliseconds: 500), _salvarDados
      );
12    Timer(Duration(seconds: 60), _finalizarColeta);
13  }
```

Fonte: Autoria própria.

de criação do diretório destinado ao armazenamento dos dados coletados pelo aplicativo. Nas linhas 11 e 12, o *timer* é ativado para iniciar a coleta de dados. Essa coleta ocorre durante 60 segundos, e a cada 0,5 segundos, a função responsável por armazenar os dados é acionada. Por fim, após os 60 segundos, a função encarregada de finalizar a coleta de dados é chamada, encerrando assim essa etapa do processo.

O Código-fonte 3 desempenha um papel importante na coleta e armazenamento dos valores dos sensores. A função acionada pela linha 11 do Código-fonte 2 é responsável por iniciar o processo de coleta de dados.

No Código-fonte 3, na linha 5, uma lista é criada para armazenar os valores dos sensores coletados. Entre as linhas 6 e 15, ocorre uma verificação para averiguar se a lista está vazia, ou seja, se nenhum dado foi adicionado anteriormente. Nesse caso, as descrições das colunas são adicionadas à lista. Na linha 16, o valor da velocidade medido pelo sensor é adicionado a uma variável chamada "data" do tipo *String*. Entre as linhas 17 e 26, são verificados os valores dos sensores de latitude, longitude, giroscópio e magnetômetro. Se esses valores não forem nulos, eles são adicionados à variável "data". Após essas etapas, na linha 28, os valores presentes na variável "data" são adicionados à lista dos valores dos sensores. Nas linhas 30 a 32, os valores coletados são salvos em um arquivo TXT e armazenados no dispositivo móvel.

Código-fonte 3 – Salvar os dados no arquivo TXT

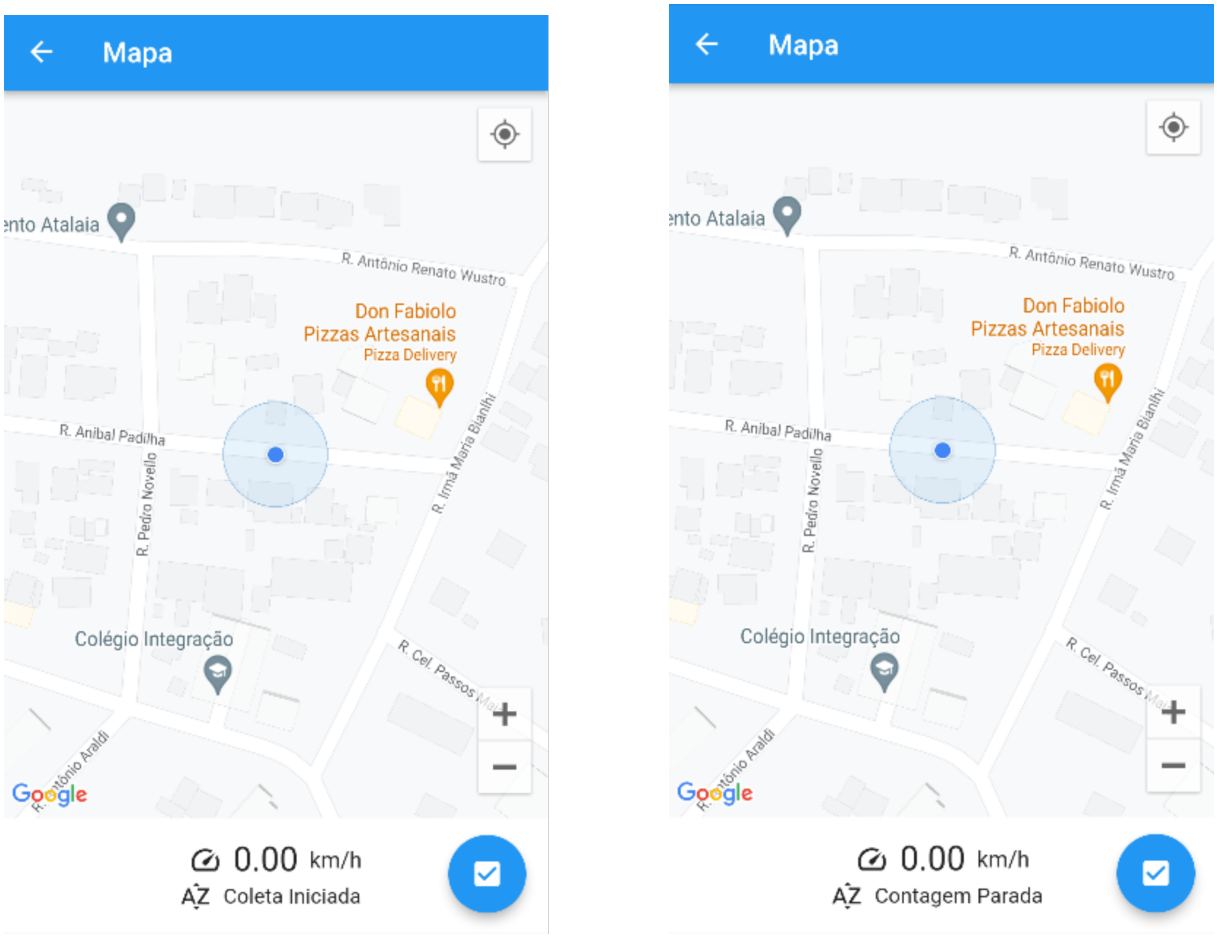
```
1
2   Future<void> _salvarDados(Timer timer) async {
3     File file = File(filePath);
4
5     List<String> listaDeValores = [];
6     try {
7       if (file.lengthSync() == 0) {
8         String data;
9         data =
10          "Velocidade, Latitude, Longitude, Giroscopio (X),
        Giroscopio (Y), Giroscopio (Z), Magnetometro
    (X), Magnetometro (Y), Magnetometro (Z)\n";
11        listaDeValores.add(data);
12        IOSink sink = file.openWrite(mode: FileMode.append);
13        sink.writeAll(listaDeValores);
14        sink.close();
15      }
16    String data = "$_velocidadeAtual";
17    if (posicaoAtual != null) {
18      data += ", ${posicaoAtual?.latitude},
        ${posicaoAtual?.longitude}";
19    }
20    if (_giroscopio != null) {
21      data += ", ${_giroscopio?.x}, ${_giroscopio?.y},
        ${_giroscopio?.z}";
22    }
23    if (_magnetometro != null) {
24      data +=
25          ", ${_magnetometro?.x}, ${_magnetometro?.y},
          ${_magnetometro?.z}\n";
26    }
27
28    listaDeValores.add(data);
29
30    IOSink sink = file.openWrite(mode: FileMode.append);
31    sink.writeAll(listaDeValores, '\n');
32    sink.close();
33  }
```

Fonte: Autoria própria.

3.2.1.1 Telas do Aplicativo

O aplicativo consiste em uma única tela, a qual é possível verificar a localização e a velocidade em tempo real do usuário. A tela possui também uma mensagem informando se o aplicativo ainda está coletando as informações para formar a base ou se a coleta já foi encerrada.

Quando o botão em azul for clicado, uma mensagem de Coleta Iniciada irá aparecer,



((a)) Coleta Iniciada **((b)) Coleta Parada**

Figura 11 – Tela do Aplicativo
Fonte: Autoria própria.

como pode ser visto na Figura 11 letra (a), então um cronômetro de 60 segundos será iniciado e quando terminar o tempo, a tela da Figura 11 letra (b) irá aparecer, sinalizando que a coleta foi finalizada. Para iniciar outra coleta de dados é só clicar no botão em azul novamente. Lembrando que a cada coleta iniciada o algoritmo recebe 119 linhas de dados, contendo os atributos descritos anteriormente na subseção 3.1.1.

3.2.2 Pré processamento da base de dados

Antes de realizar a etapa de treinamento dos algoritmos de aprendizagem de máquina, foi necessário realizar o pré-processamento dos dados coletados. A primeira etapa desse processo foi a rotulagem dos dados, na qual os dados coletados durante a atividade de estacionamento do veículo foram rotulados com o valor 1 na coluna "Estado" criada para esse fim, enquanto os dados coletados durante o movimento do veículo foram rotulados com o valor 0 na mesma

coluna "Estado".

Com a rotulagem concluída, realizou-se a normalização dos dados, o que é fundamental para garantir a qualidade e uniformidade dos mesmos e, consequentemente, obter resultados mais precisos e confiáveis durante o processo de treinamento. Além disso, foram excluídas as colunas de latitude e longitude da base de dados, a fim de evitar *overfitting*. Como os dados foram coletados em um único município, o modelo estatístico de treinamento poderia se ajustar ao conjunto de dados. Após o pré-processamento, obteve-se a base de dados representada na Tabela 1. Para comparar os resultados de treinamento, optou-se por separar cada coleta em blocos de 119 linhas e transformar essas linhas em colunas, como pode ser verificado na nova base formada, representada na Tabela 2. Essa nova base permite verificar qual estratégia resulta em um melhor resultado na etapa de treinamento do algoritmo de classificação.

3.2.3 Algoritmos de classificação

Os seguintes algoritmos de classificação foram utilizados na realização do treinamento da base de dados obtida, sendo eles, k-NN, *Random Forest* e por fim o algoritmo SVM .

3.2.3.1 *k-Nearest Neighbors*

Segundo (SONI, 2018) o método *k-Nearest Neighbors* (*k-NN*) é um algoritmo simples de aprendizado de máquina que pode ser utilizado para resolver problemas de classificação. Sua implementação é baseada no princípio de que objetos semelhantes estão próximos uns dos outros no espaço de características. O algoritmo não requer uma fase de treinamento explícita, pois os dados de treinamento são usados diretamente durante a classificação. Para a realização do treinamento, a implementação do algoritmo foi realizada no *Google Colab* utilizando a linguagem de programação *Python*, como é possível visualizar no Código-fonte 4.

O Código-fonte 4 inicia importando as principais bibliotecas necessárias para a execução do código, como o *pandas*, *numpy* e *scikit-learn*. Em seguida, nas linhas 3 e 4, é realizada a importação da base de dados utilizada no treinamento do algoritmo, sendo que a leitura dos dados é feita diretamente do Google Drive.

Após a importação da base de dados, nas linhas 5 e 7, ocorre a separação das variáveis preditoras e a variável objetivo (*target*). Nessa etapa, os dados da base de dados são divididos em duas partes: X e y. A variável X contém as variáveis preditoras, ou seja, as características que

Código-fonte 4 – Implementação do *k-NN*

```
1 #Importação das bibliotecas#
2 #Importação da base de dados
3 drive.mount('/content/drive')
4 df = pd.read_csv('/content/drive/My Drive/Bases TCC Transpostas/
    BaseTranspostaCompleta.csv')
5 # Separar as variáveis preditoras e a variável target
6 X = df.iloc[:, :-1].values # variáveis preditoras
7 y = df.iloc[:, -1].values # variável target
8 # Normalizar as variáveis preditoras
9 scaler = StandardScaler()
10 X = scaler.fit_transform(X)
11 # Dividir a base de dados em conjuntos de treinamento e teste
12 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.4, random_state=0)
13 # Criar um modelo de classificação k-NN
14 knn = KNeighborsClassifier(n_neighbors=5)
15 # Treinar o modelo com os dados de treinamento
16 knn.fit(X_train, y_train)
17 # Fazer previsões com os dados de teste
18 y_pred = knn.predict(X_test)
19 # Calcular a acurácia do modelo
20 accuracy = knn.score(X_test, y_test)
21 print('Acurácia do modelo: {:.2f}%'.format(accuracy*100))
```

Fonte: Autoria própria.

serão usadas para prever a classe do veículo, enquanto a variável *y* contém a classe do veículo.

Em seguida, nas linhas 8 a 10, é realizada a normalização das variáveis preditoras. Isso é feito utilizando a classe StandardScaler da biblioteca *scikit-learn*. A normalização é importante para garantir que as variáveis estejam na mesma escala, evitando que alguma variável tenha um peso maior do que as outras durante o treinamento do modelo.

Na linha 12, a base de dados é dividida em conjuntos de treinamento e teste usando a função *train_test_split* da biblioteca *scikit-learn*. Nesse exemplo, 40% dos dados são reservados para teste, enquanto 60% são utilizados para treinamento do modelo.

Após a normalização e a divisão da base de dados em conjuntos de treinamento e teste, na linha 14, é criado um objeto do tipo *KNeighborsClassifier* da biblioteca *scikit-learn*. O parâmetro *n_neighbors* define o número de vizinhos mais próximos a serem considerados para a classificação. Nesse caso, o valor é definido como 5.

Na linha 16, o modelo K-NN é treinado utilizando o método *fit*, onde são passados os conjuntos de treinamento *X_train* e *y_train* como argumentos. Durante o treinamento, o modelo aprende a relação entre as variáveis preditoras e a variável objetivo.

Após o treinamento, na linha 18, as previsões são feitas utilizando o método *predict* do modelo K-NN. É passado o conjunto de teste *X_test* como argumento, e o modelo utiliza os vizinhos mais próximos para atribuir uma classe a cada instância de teste.

Na linha 20, a acurácia do modelo é calculada usando o método score do modelo K-NN. São passados como argumentos os conjuntos de teste *X_test* e *y_test*. A acurácia é uma métrica que mede a proporção de previsões corretas em relação ao total de previsões. Por fim, na linha 21, o valor da acurácia formatada em porcentagem é impresso na tela.

3.2.3.2 *Random Forest*

Segundo (CHEN; ANDY; BREIMAN, 2004) o *Random Forest* é um conjunto de árvores de decisão para classificação ou regressão. Ela é construída a partir de amostras de *bootstrap* dos dados de treinamento e utiliza seleção aleatória das características durante o processo de construção das árvores. A previsão é feita combinando as previsões individuais das árvores por meio de votação majoritária para classificação, ou média para regressão. A implementação do algoritmo foi realizada utilizando a linguagem de programação *Python* no ambiente do *Google Colab*, como pode ser observado no código a seguir, representado pelo Código-fonte 5.

Código-fonte 5 – Implementação do *Random Forest*

```
1 #Importação da base de dados
2 drive.mount('/content/drive')
3 df = pd.read_csv('/content/drive/My Drive/Bases TCC Transpostas/
    BaseTranspostaCompleta.csv')
4 # Separar as variáveis preditoras e a variável target
5 X = df.drop('Estado', axis=1)
6 y = df['Estado']
7 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.4, random_state=42)
8 # Normalizar as variáveis preditoras
9 scaler = StandardScaler()
10 scaler.fit(X_train)
11 X_train_scaled = scaler.transform(X_train)
12 X_test_scaled = scaler.transform(X_test)
13 # Realizar o treinamento utilizando o Random Forest
14 rfc = RandomForestClassifier(n_estimators=100)
15 rfc.fit(X_train_scaled, y_train)
16 y_pred = rfc.predict(X_test_scaled)
17 # Calcular a acurácia do modelo
18 accuracy = accuracy_score(y_test, y_pred)
19 print("Acurácia:", accuracy)
```

Fonte: Autoria própria.

O Código-fonte 5 inicia importando as principais bibliotecas necessárias para a execução do código, como *pandas*, *numpy* e *scikit-learn*. Em seguida, nas linhas 2 e 3, é feita a importação da base de dados utilizada no treinamento do algoritmo, sendo que os dados são lidos diretamente do *Google Drive*.

Nas linhas 5 a 6, os dados da base de dados são divididos em duas partes: *X* e *y*. A variável *X* contém as variáveis preditoras, todas as colunas da base de dados, exceto a coluna "Estado". Já a variável *y* contém a variável alvo, que é a classe do veículo, representada pela coluna "Estado".

Na linha 7, a base de dados é dividida em conjuntos de treinamento e teste usando a função *train_test _split* do *scikit-learn*. Neste exemplo, 40% dos dados são reservados para teste, enquanto 60% são usados para treinamento.

Entre as linhas 9 e 12, os dados são normalizados. Essa etapa é importante para garantir que as variáveis estejam na mesma escala e evitar que alguma variável tenha um peso maior do que as outras durante o treinamento do modelo.

Na linha 14, é criado um objeto do tipo *RandomForestClassifier* do *scikit-learn*, com 100 estimadores (árvores de decisão). Em seguida, na linha 15, o modelo é treinado utilizando o método *fit*, no qual são passadas as variáveis preditoras normalizadas (*X_train_scaled*) e a variável alvo (*y_train*) como argumentos. Durante o treinamento, o modelo aprende a relação entre as variáveis preditoras e a variável alvo.

Na linha 16, as previsões são feitas utilizando o método *predict* do modelo *Random Forest*, passando as variáveis preditoras normalizadas dos dados de teste (*X_test_scaled*) como argumento. O modelo utiliza as árvores de decisão para atribuir uma classe a cada instância de teste.

Na linha 18, a acurácia do modelo é calculada utilizando a função *accuracy_score* do *scikit-learn*. São passados como argumentos a variável alvo dos dados de teste (*y_test*) e as previsões do modelo (*y_pred*). A acurácia é uma métrica que mede a proporção de previsões corretas em relação ao total de previsões. Por fim, o valor da acurácia é apresentado na tela.

3.2.3.3 *Support Vector Machine*

Segundo (SANJAA; CHULUUN, 2013) o *SVM*, *Support Vector Machine*, é uma técnica utilizada para a classificação de dados. No processo de classificação, o objetivo do *SVM* é criar um modelo capaz de prever a classificação de um elemento do conjunto de teste com base

nas características fornecidas. O algoritmo pode empregar diferentes tipos de kernels, sendo os principais o linear, polinomial, RBF (*Radial Basis Function*) e *sigmoid*. O *SVM* linear é especialmente adequado para conjuntos de dados massivos que possuem muitas características. Em contraste, o *SVM* não linear tende a ter resultados inadequados ao lidar com conjuntos de dados que possuem mais de 10.000 entradas. O algoritmo foi implementado utilizando a linguagem de programação Python no ambiente do *Google Colab*, como exemplificado no Código-fonte 6.

O Código-fonte 6 inicia importando as principais bibliotecas necessárias para a execução do código. Em seguida, nas linhas 2 e 3, é realizada a leitura da base de dados que estava armazenada no *Google Drive*.

Código-fonte 6 – Implementação do *SVM*

```
1 #Importação da base de dados
2 drive.mount('/content/drive')
3 df = pd.read_csv('/content/drive/My Drive/Bases TCC Transpostas/
    BaseTranspostaCompleta.csv')
4 # Separar os dados em treinamento e teste
5 X = df.iloc[:, :-1].values
6 y = df.iloc[:, -1].values
7 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.4, random_state=42)
8 # Normalizar os dados
9 scaler = StandardScaler()
10 X_train = scaler.fit_transform(X_train)
11 X_test = scaler.transform(X_test)
12 # Treinar o modelo SVM
13 svm = SVC(kernel='linear')
14 svm.fit(X_train, y_train)
15 # Avaliar o modelo SVM
16 y_pred = svm.predict(X_test)
17 accuracy = np.mean(y_pred == y_test)
18 print('Acurácia: {:.2f}%'.format(accuracy * 100))
```

Fonte: Autoria própria.

Nas linhas 5 e 6, ocorre a divisão dos dados da base de treinamento em duas partes: *X* e *y*. A variável *X* contém todas as colunas da base de dados, exceto a última, que representa a variável *target*. Já a variável *y* armazena a última coluna, que é a variável *target*.

Na linha 7, os dados são divididos em conjuntos de treinamento e teste utilizando a função *train_test_split* da biblioteca *scikit-learn*. Neste exemplo, 40% dos dados são destinados para teste, enquanto 60% são usados para treinamento. O parâmetro *random_state* é utilizado para definir a semente para a geração dos números aleatórios, garantindo a reprodutibilidade dos

resultados.

As linhas 9, 10 e 11 são responsáveis pela normalização dos dados. Essa etapa é importante para garantir que as variáveis estejam na mesma escala.

Na linha 13, é criado um objeto do tipo SVC, *Support Vector Classifier* da biblioteca *scikit-learn*, com o *kernel* linear. O modelo SVM é treinado utilizando o método *fit*, no qual são passados os dados de treinamento normalizados (*X_train*) e a variável target de treinamento (*y_train*) como argumentos. Durante o treinamento, o modelo SVM utiliza as informações para aprender a classificar as instâncias de teste.

Na linha 16, as previsões são feitas utilizando o método predict do modelo SVM, passando os dados de teste normalizados (*X_test*) como argumento. O modelo utiliza as informações aprendidas durante o treinamento para atribuir uma classe a cada instância de teste.

A acurácia do modelo é calculada na linha 17, comparando as previsões (*y_pred*) com a variável target de teste (*y_test*). A função *np.mean* é utilizada para calcular a média das correspondências corretas entre as previsões e os valores reais de teste. Por fim, o valor da acurácia é impresso na tela formatado em porcentagem.

3.2.4 Medidas

Para os testes de desempenho dos modelos apresentados, foi utilizado a medida de desempenho *Accuracy*. Segundo (AMBASHT, 2020), *accuracy* é uma medida de desempenho a qual é calculada como a razão entre "todas as predições realizadas corretamente (*True Positive* (TP) e *True Negative* (TN))"e "todas as predições realizadas, incluindo todas as amostras (*False Positive (FP) e False Negative (FN)*)". A *accuracy* é capaz de medir a porcentagem de vezes que um modelo de aprendizado de máquina ou algoritmo classificou corretamente um determinado conjunto de teste. Essa medida de desempenho é representada pela seguinte equação:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{8}$$

4 RESULTADOS

Neste trabalho, como já mencionado no Capítulo 1, o objetivo principal é desenvolver uma análise e estudo a respeito da possibilidade de distinguir, por meio de algoritmos de aprendizagem de máquina, as atividades de estacionar e estar em movimento de um veículo. Para isso, foram utilizados três modelos de aprendizado de máquina: k-NN, Random Forest e SVM. A etapa de treinamento dos algoritmos foi dividida em duas fases. Na primeira fase, foram utilizadas as bases de dados mencionadas na Subseção 3.1.1. Na segunda fase, algumas colunas dessas bases foram excluídas.

Nesta seção, apresentaremos os principais resultados obtidos, além dos atributos que tiveram maior influência nos resultados.

4.1 PRÉ-PROCESSAMENTO

Para executar a segunda fase de testes, foi necessário excluir as colunas do sensor de giroscópio (x,y e z) da base de dados mencionada na Tabela 1. Inicialmente, essa base de dados possuía um total de 8 colunas. Após a exclusão dos dados do sensor de giroscópio, restaram 5 colunas, mantendo as 11.194 linhas.

Quanto à base de dados representada pela Tabela 2 em que os dados foram separados em blocos de 119 linhas e essas linhas foram transformadas em colunas, inicialmente, tínhamos um total de 834 colunas. Após a exclusão das colunas do sensor de giroscópio, restaram 477 colunas. Essa diferença significativa ocorreu porque, em cada linha da base, havia colunas correspondentes ao giroscopio_x_1, giroscopio_y_1, giroscopio_z_1 ... giroscopio_x_119, giroscopio_y_119, giroscopio_z_119.

4.2 AVALIAÇÃO DOS MODELOS

Durante a primeira fase de avaliação dos modelos, foram utilizadas duas bases de dados chamadas Base 1 e Base 2. Essas bases foram incorporadas nos treinamentos dos algoritmos de classificação, incluindo as colunas referentes aos dados do giroscópio. Na segunda fase de avaliação, os dados relacionados ao sensor do giroscópio foram removidos das bases de dados.

Para garantir a formação adequada dos algoritmos, foram realizadas comparações entre os modelos em dois softwares distintos. Os resultados apresentados nas Tabelas 3 e 4 representam

a acurácia alcançada no ambiente do Google Colab, utilizando a linguagem de programação Python. Durante a execução dos algoritmos, a divisão das bases foi feita da seguinte maneira: 40% dos dados foram reservados para a base de teste, enquanto 60% foram utilizados para o treinamento.

Adicionalmente, os mesmos conjuntos de dados e algoritmos de classificação foram treinados no WEKA, resultando em uma acurácia similar àquela obtida no Google Colab.

Tabela 3 – Acurácia medida dos modelos na Fase 1

Base de treinamento	*k-NN*	*Random Forest*	*SVM*
Base 1	90,04	92,71	61,30
Base 2	75,68	83,78	81,08

Fonte: Autoria própria.

A Tabela 3 apresenta os resultados finais da primeira fase de treinamento. Nessa fase, foram utilizadas duas bases de treinamento, as quais são mencionadas na Subseção 3.1.1: Base 1, com 11.194 linhas, e Base 2, com 92 linhas. Os algoritmos de classificação aplicados foram k-NN, Random Forest e SVM.

Na Base 1, os algoritmos demonstraram desempenhos distintos. O k-NN obteve uma acurácia de 90,04% e, por meio de testes de correlação das características, identificou-se que os atributos mais influentes na classificação foram, em ordem, "Magnetômetro (X)", "Magnetômetro (Y)", "Giroscópio (Y)", "Velocidade"e "Giroscópio (X)". O algoritmo *Random Forest* alcançou uma acurácia de 92,71% e os atributos mais relevantes para a classificação foram, em ordem, "Velocidade", "Magnetômetro (Y)", "Magnetômetro (X)"e "Magnetômetro (Z)". No caso do SVM, que obteve uma acurácia de 61,30%, não é possível identificar diretamente as características mais importantes como nos casos anteriores. O desempenho inferior do SVM pode ser atribuído à sua melhor adequação a bases de dados lineares com menos atributos. Dessa forma, devido à natureza não linear e ao grande número de itens presentes na Base 1, o SVM apresentou um desempenho de classificação inferior em comparação aos outros algoritmos.

Na Base 2, os algoritmos apresentaram diferentes desempenhos. O k-NN obteve uma acurácia de 75,68% e os atributos mais influentes na classificação foram identificados como giroscópio_x_3, giroscópio_x_2, giroscópio_y_2 e giroscópio_z_1. O algoritmo *Random Forest* alcançou uma acurácia de 83,78% e os atributos mais relevantes para a classificação foram velocidade_119, velocidade_115, velocidade_117, velocidade _116 e velocidade _102. Já o algoritmo SVM obteve uma acurácia de 81,08%, porém não foi possível identificar diretamente as características mais importantes, como nos casos anteriores. É interessante notar que, mesmo

com a Base 2 contendo 834 colunas de características coletadas, o algoritmo SVM obteve uma acurácia consideravelmente melhor em comparação com a classificação da Base 1.

Analisando a primeira fase de testes, o algoritmo que apresentou o melhor desempenho na classificação das atividades foi o *Random Forest* utilizando a Base 1, com uma acurácia de 92,71%.

Tabela 4 – Acurácia medida dos modelos na Fase 2

Base de treinamento	*k-NN*	*Random Forest*	*SVM*
Base 1	92,83	94,12	59,98
Base 2	81,08	86,48	78,38

Fonte: Autoria própria.

Os resultados da segunda fase de treinamento estão dispostos na Tabela 4. Nessa etapa, foram utilizadas as mesmas Bases 1 e 2 da fase anterior, porém, excluindo as colunas referentes ao giroscópio. Novamente, os algoritmos k-NN, *Random Forest* e SVM foram aplicados.

Na segunda fase, com a Base 1, os algoritmos apresentaram os seguintes desempenhos. O k-NN alcançou uma acurácia de 92,83% e os atributos mais influentes foram "Velocidade", "Magnetômetro (X)"e "Magnetômetro (Y)". O *Random Forest* obteve uma acurácia de 94,12% e os atributos mais influentes foram "Velocidade", "Magnetômetro (Y)"e "Magnetômetro (Z)". Já o SVM obteve uma acurácia de 59,98% e não foi possível identificar diretamente as características mais importantes. É importante destacar que o SVM teve o pior desempenho, como também observado na primeira fase.

Na segunda fase com a Base 2, os algoritmos apresentaram os seguintes desempenhos. O k-NN obteve uma acurácia de 81,08% e os atributos mais influentes foram "magnetometro_y_2", "magnetometro_x_2", "magnetometro_z_2", "magnetometro_y_4"e "velocidade_1". O *Random Forest* obteve uma acurácia de 86,48% e os atributos mais influentes foram "velocidade_115", "velocidade_114", "velocidade_119", "velocidade_118". O SVM obteve uma acurácia de 78,38% e, assim como na fase anterior, não foi possível identificar diretamente as características mais importantes. Mais uma vez, o *Random Forest* obteve o melhor desempenho na classificação das atividades, alcançando uma acurácia de 94,12%.

Em resumo, o algoritmo *Random Forest* utilizando a Base 1 obteve a melhor classificação em ambas as fases, sendo que na segunda fase houve um aumento de 1,41% na acurácia em comparação com a primeira fase.

5 CONCLUSÃO

Após a conclusão deste projeto, foi possível alcançar o objetivo de distinguir com precisão as atividades relacionadas ao estacionamento de um veículo por meio de algoritmos de classificação. A análise do desempenho dos modelos em duas fases de treinamento, utilizando diferentes bases de dados, resultou em resultados promissores.

Na primeira fase de treinamento, utilizando a Base 1, o algoritmo *Random Forest* obteve uma acurácia de 92,71%, classificando corretamente 92,71% das instâncias de atividades de estacionamento. O algoritmo k-NN alcançou uma acurácia de 90,04%, enquanto o SVM apresentou um desempenho inferior, com uma acurácia de 61,30%. Esses resultados indicam que o SVM não se adaptou bem ao contexto não linear e à grande quantidade de elementos presentes na Base 1.

Na segunda fase de treinamento, após a remoção dos atributos relacionados ao sensor de giroscópio nas bases de dados, o algoritmo *Random Forest* apresentou um desempenho ainda melhor, atingindo uma acurácia de 94,12% na Base 1. O k-NN obteve uma acurácia de 92,83%, enquanto o SVM teve a pior performance, com uma acurácia de 59,98%.

Ao analisar a Base 2, mesmo com um número maior de colunas (834), o algoritmo SVM alcançou uma acurácia de 81,08% na primeira fase e 78,38% na segunda fase. Por outro lado, o *Random Forest* obteve uma acurácia de 83,78% na primeira fase e 86,48% na segunda fase. Esses resultados destacam a capacidade do algoritmo *Random Forest* em lidar com conjuntos de dados complexos e obter um melhor desempenho de classificação em comparação com os demais algoritmos.

Em resumo, conclui-se que o algoritmo *Random Forest*, utilizando a Base 1, foi o mais eficaz na classificação das atividades de estacionamento, com uma acurácia de 92,71% na primeira fase e 94,12% na segunda fase. Esses resultados quantificam o desempenho do modelo e evidenciam sua capacidade de distinguir com precisão as atividades de estacionamento em diferentes contextos.

5.1 TRABALHOS FUTUROS

Como perspectivas para trabalhos futuros, recomenda-se explorar outras técnicas de pré-processamento de dados visando aprimorar o desempenho dos modelos de classificação. Além disso, considera-se fundamental o aumento do tamanho da base de dados, pois uma

amostra mais representativa proporciona resultados mais confiáveis. A expansão da base de dados permitirá uma análise mais sólida do desempenho dos modelos de classificação, conferindo maior robustez e confiabilidade às conclusões.

Uma abordagem adicional é aprimorar a diversidade de cenários e contextos nos testes realizados. Para avaliar a robustez e a generalização dos modelos de classificação, é essencial coletar dados em diferentes cenários, abrangendo ambientes distintos, modelos de veículos variados, cidades e usuários diferentes. A diversidade de cenários contribuirá para uma análise mais abrangente do desempenho dos modelos, permitindo verificar sua adaptabilidade a diferentes contextos e proporcionando uma compreensão mais completa de sua eficácia.

Esses resultados têm potencial de aplicação em um possível aplicativo de auxílio ao estacionamento. Com o modelo de classificação treinado pelo algoritmo *Random Forest*, é possível implementar uma funcionalidade que identifica automaticamente as atividades de estacionamento, oferecendo informações relevantes aos usuários, como disponibilidade de vagas, direcionamento para estacionamentos próximos, estatísticas sobre padrões de estacionamento e também apresentar ao usuário a última localização de onde o veículo foi estacionado. Isso proporcionaria uma experiência mais conveniente e eficiente para os motoristas, contribuindo para a otimização do processo de estacionamento.

REFERÊNCIAS

ALI, M. E. Global Positioning System (GPS): Definition, Principles, Errors, Applications & DGPS. **no. April**, 2020.

ALPAYDIN, E. **Introduction to machine learning**. [*S. l.*]: MIT press, 2020.

AMBASHT, K. **Understanding Precision, Recall, and Accuracy with COVID tests**. 2020. Disponível em: https://towardsdatascience.com/understanding-precision-recall-and-accuracy-with-covid-tests-c1c9e93bc92a. Acesso em: 7 mai. 2023.

ANDROID. **Arquitetura da Plataforma**. 2020. Disponível em: https://developer.android.com/guide/platform. Acesso em: 10 mai. 2022.

ARPACI-DUSSEAU, R. H.; ARPACI-DUSSEAU, A. C. **Operating systems: Three easy pieces**. [*S. l.*]: Arpaci-Dusseau Books LLC Boston, 2018.

BASAVARAJU, P.; VARDE, A. S. Supervised learning techniques in mobile device apps for Androids. **ACM SIGKDD Explorations Newsletter**, ACM New York, NY, USA, v. 18, n. 2, p. 18–29, 2017.

CHEN, C.; ANDY, L.; BREIMAN, L. **Using Random Forest to Learn Imbalanced Data**. 2004. Disponível em: https://towardsdatascience.com/introduction-to-k-nearest-neighbors-3b534bb11d26. Acesso em: 10 mai. 2023.

DEVROYE, L.; GYÖRFI, L.; LUGOSI, G. Consistency of the k-nearest neighbor rule. In: A Probabilistic Theory of Pattern Recognition. [*S. l.*]: Springer, 1996. P. 169–185.

DJUKNIC, G. M.; RICHTON, R. E. Geolocation and assisted GPS. **Computer**, IEEE, v. 34, n. 2, p. 123–125, 2001.

EMBRAPA. **GPS - Global Positioning System**. 2020. Disponível em: https://www.embrapa.br/satelites-de-monitoramento/missoes/gps. Acesso em: 4 jun. 2023.

FACELI, K. *et al.* Inteligência artificial: uma abordagem de aprendizado de máquina, 2011.

GAO, C. *et al.* A study of mobile device utilization. In: IEEE. 2015 ieee international symposium on performance analysis of systems and software (ispass). [*S. l.: s. n.*], 2015. P. 225–234.

GENTILE, C. *et al.* **Geolocation techniques: principles and applications**. [*S. l.*]: Springer Science & Business Media, 2012.

GUSTAFSSON, F. Geolocation: Maps, measurements and methods. In: IET. 9TH IET Data Fusion & Target Tracking Conference (DF&TT 2012): Algorithms & Applications. [*S. l.: s. n.*], 2012. P. 1–48.

KAPLAN, E. D.; HEGARTY, C. J. **Understanding GPS: Principles and Applications, Norwood, MA: Artech House**. [*S. l.*]: Inc, 2006.

LECHETA, R. **Android Essencial: Edição resumida do livro Google Android**. [*S. l.*]: Novatec Editora, 2016.

LECHETA, R. **Google Android 4ª edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. [*S. l.*]: Novatec Editora, 2015.

MARWEDEL, P. Embedded system hardware. In: EMBEDDED System Design. [*S. l.*]: Springer, 2021. P. 127–201.

MITCHELL, T. M.; MITCHELL, T. M. **Machine learning**. [*S. l.*]: McGraw-hill New York, 1997. v. 1.

NOVAC, O. C. *et al.* Comparative study of Google Android, Apple iOS and Microsoft Windows phone mobile operating systems. In: IEEE. 2017 14th international conference on engineering of modern electric systems (EMES). [*S. l.: s. n.*], 2017. P. 154–159.

SANJAA, B.; CHULUUN, E. Malware detection using linear SVM. In: IFOST. [*S. l.: s. n.*], 2013. v. 2, p. 136–138. DOI: 10.1109/IFOST.2013.6616872.

SHEIKH, A. A. *et al.* Smartphone: Android Vs IOS. **The SIJ Transactions on Computer Science Engineering & its Applications (CSEA)**, v. 1, n. 4, p. 141–148, 2013.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating system concepts**. [*S. l.*]: John Wiley & Sons, 2006.

SILVA, J. R. Geolocalização GPS para Dispositivos Móveis: Concepção De Uma Ferramenta Para Coleta De Dados Georreferenciados, p. 77, 2014.

SONI, D. **Introduction to k-Nearest-Neighbors**. 2018. Disponível em: https://towardsdatascience.com/introduction-to-k-nearest-neighbors-3b534bb11d26. Acesso em: 10 mai. 2023.

TANENBAUM, A.; BOS, H. **Modern operating systems**. [*S. l.*]: Pearson Education, Inc., 2014.

TRACY, K. W. Mobile application development experiences on Apple's iOS and Android OS. **Ieee Potentials**, IEEE, v. 31, n. 4, p. 30–34, 2012.

WINDMILL, E. **Flutter in action**. [*S. l.*]: Simon e Schuster, 2020.

ANEXO A — REPOSITÓRIO DO TRABALHO

O repositório no GitHub contendo os códigos na íntegra, juntamente com a base de dados e os algoritmos de aprendizagem de máquina estão disponíveis em: https://github.com/ViniFrandoloso/TCC