Trabalho de Conclusão de Curso

Submetido em 2024

Aplicação de *Machine Learning* para Identificação e Distinção de Movimentos da Mão Baseados em Dados Eletromiográficos do Antebraço

Autores

Bárbara Nery de Souza

Vitor Ferreira Paschoal

Departamento de Engenharia Elétrica e de Computação - EESC/USP

Orientador

Profº Dr. Alberto Cliquet Junior

Departamento de Engenharia Elétrica e de Computação - EESC/USP

São Carlos, SP

Novembro de 2024

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues
Fontes da EESC/USP

S729a

Souza, Bárbara Nery de
Aplicação de *machine learning* para identificação e distinção de movimentos da mão baseados em dados eletromiográficos do antebraço / Bárbara Nery de Souza, Vitor Ferreira Paschoal; orientador Alberto Cliquet Junior. -- São Carlos, 2024.

Monografia (Graduação em Engenharia Elétrica com Ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2024.

1. Inteligência artificial. 2. Classes. 3. Classificação. 4. Movimentos. 5. Acuracia. 6. Precisão. 7.Matriz de confusão. I. Paschoal, Vitor Ferreira. II. Titulo.

Elaborado por Elena Luzia Palloni Gonçalves – CRB 8/4464

FOLHA DE APROVAÇÃO

Nome: Bárbara Nery de Souza

Título: "Aplicação de Machine Learning para Identificação e Distinção de Movimentos da Mão Baseados em Dados Eletromiográficos do Antebraço"

Trabalho de Conclusão de Curso defendido e aprovado em 21 / 11 / 2024 **,**

com NOTA 9,5 (nove, cinco) **, pela Comissão Julgadora:**

Prof. Titular Alberto Cliquet Júnior - Orientador - SEL/EESC/USP

Dr. Orivaldo Lopes da Silva - EESC/USP

Prof. Titular Ivan Nunes da Silva - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Professor Associado José Carlos de Melo Vieira Júnior

FOLHA DE APROVAÇÃO

Nome: Vitor Ferreira Paschoal

Título: "Aplicação de Machine Learning para Identificação e Distinção de Movimentos da Mão Baseados em Dados Eletromiográficos do Antebraço"

Trabalho de Conclusão de Curso defendido e aprovado em 21/11/2024,

com NOTA 9,5 (nove, cinco), pela Comissão Julgadora:

Prof. Titular Alberto Cliquet Júnior - Orientador - SEL/EESC/USP

Dr. Orivaldo Lopes da Silva - EESC/USP

Prof. Titular Ivan Nunes da Silva - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Professor Associado José Carlos de Melo Vieira Júnior

Agradecimentos

Eu, Vitor Paschoal, gostaria de agradecer, antes de tudo, à minha família, Carlos, Silvia e Mariana. Sou muito grato por todo apoio que me deram, todas as dificuldades que passaram para me fazer chegar onde estou agora e por sempre estarem comigo em cada passo da minha caminhada. Agradeço à minha namorada Isadora por ter me ajudado sempre que precisava e por me apoiar a cada dia. Sou muito feliz por ter vocês em minha vida e tenho certeza que se não fosse por ter todos ao meu lado seria mais difícil chegar aqui. Agradeço à Deus por ter me dado saúde e entendimento para poder desenvolver esse trabalho. E, por fim, agradeço ao professor Alberto Cliquet por fazer com que esse trabalho se tornasse possível.

Eu, Bárbara Nery, agradeço à minha família (Edla, Debora, Aírto e Rodney) e aos meus amigos Harumy, Marielle, Beatriz, Vitor e Norberto por todo suporte e amor; agradeço por, em momentos diferentes, terem tido o zelo de me colocar em pé no caminho da vida. Agradeço à mãe USP pelo presente que foram esses intensos anos de engenharia e vivência universitária na história dessa sertaneja peregrina, curiosa demais para se limitar geograficamente. E, por fim, agradeço ao professor Alberto Cliquet pelas cuidadosas orientações e esforço constante em despertar nos alunos o amor pela área.

Sumário

Lista de Figuras

Nomenclatura

Ca^+ Íons de cálcio

K^+ Íons de potássio

Na^+ Íons de sódio

Ag Prata

AgCl Cloreto de prata

AGI Inteligência Artificial Geral

ANI Inteligência Artificial Estreita

ASI Superinteligência Artificial

DT Árvore de decisão

EMG Eletromiograma

IA Inteligência Artificial

K-NN *K-Nearest Neighbor*

MLP *Multi-Layer Perceptron*

MUAPT *Motor Unit Action Potential Train*

OS Operating System

PCA *Principal Component Analysis*

SVM *Support Vector Machine*

Resumo

Este documento relata os passos tomados durante o desenvolvimento de um *Multi-Layer Perceptron* (MLP) para treinar uma inteligência artificial (IA) especializada na identificação de gestos da mão utilizando sinais eletromiográficos dos músculos do antebraço. Contudo, o projeto também explorou a qualidade da base de dados utilizada, considerando características anatômicas, que guiaram o processo. Para essa análise, empregou-se algoritmos de aprendizado não supervisionado, como o K-means, na criação de um gráfico do método do cotovelo e para a criação de *clusters* para avaliar as distâncias entre eles. Ainda, utilizou-se algoritmos de aprendizado supervisionado, como K-NN, árvore de decisão e *Random Forest*. Além disso, para melhorar a visibilidade dos dados e resultados dos treinamentos, foram desenvolvidos códigos para a plotagem de mapas de dispersão e matrizes de confusão multi-classe, respectivamente. Desse modo, como resultado, ao final da avaliação dos dados e de sua filtragem, foi possível atingir um modelo de identificação com acurácia de 83,36%, precisão de 83,75%, *recall* de 83,36% e *F1 Score* de 83,41%.

Palavras-chave: inteligência artificial, acurácia, precisão, classes, movimento, EMG, MLP

1 Introdução Médica

1.1 Células Neuronais e Contração Muscular

Neurônios (ou células nervosas) são os blocos de construção do cérebro. Embora tenham os mesmos genes, a mesma organização geral e o mesmo aparato bioquímico que outras células, eles também têm características únicas que fazem o cérebro funcionar de uma maneira singular. As especializações importantes do neurônio incluem um formato de célula distinto, uma membrana externa semipermeável capaz de gerar impulsos nervosos e uma estrutura única capaz de gerar sinapses para transferir informações de um neurônio para o outro [2].

Os neurônios possuem três partes principais, cada uma com funções específicas: o corpo celular (ou soma), os dendritos e o axônio. O corpo celular abriga o citoplasma, o núcleo e as organelas [8].

O processo de contração muscular é desencadeado pelo potencial de ação gerado pelo neurônio motor, resultando em potenciais de ação nas fibras musculares. Esses potenciais, quando somados, formam o sinal eletromiográfico que leva à contração muscular.

Se tratando dos neurônios motores (responsáveis por controlar diretamente a contração das fibras musculares) o corpo celular está situado entre os dendritos e o axônio, já em alguns neurônios sensoriais, está localizado discretamente na borda do axônio.

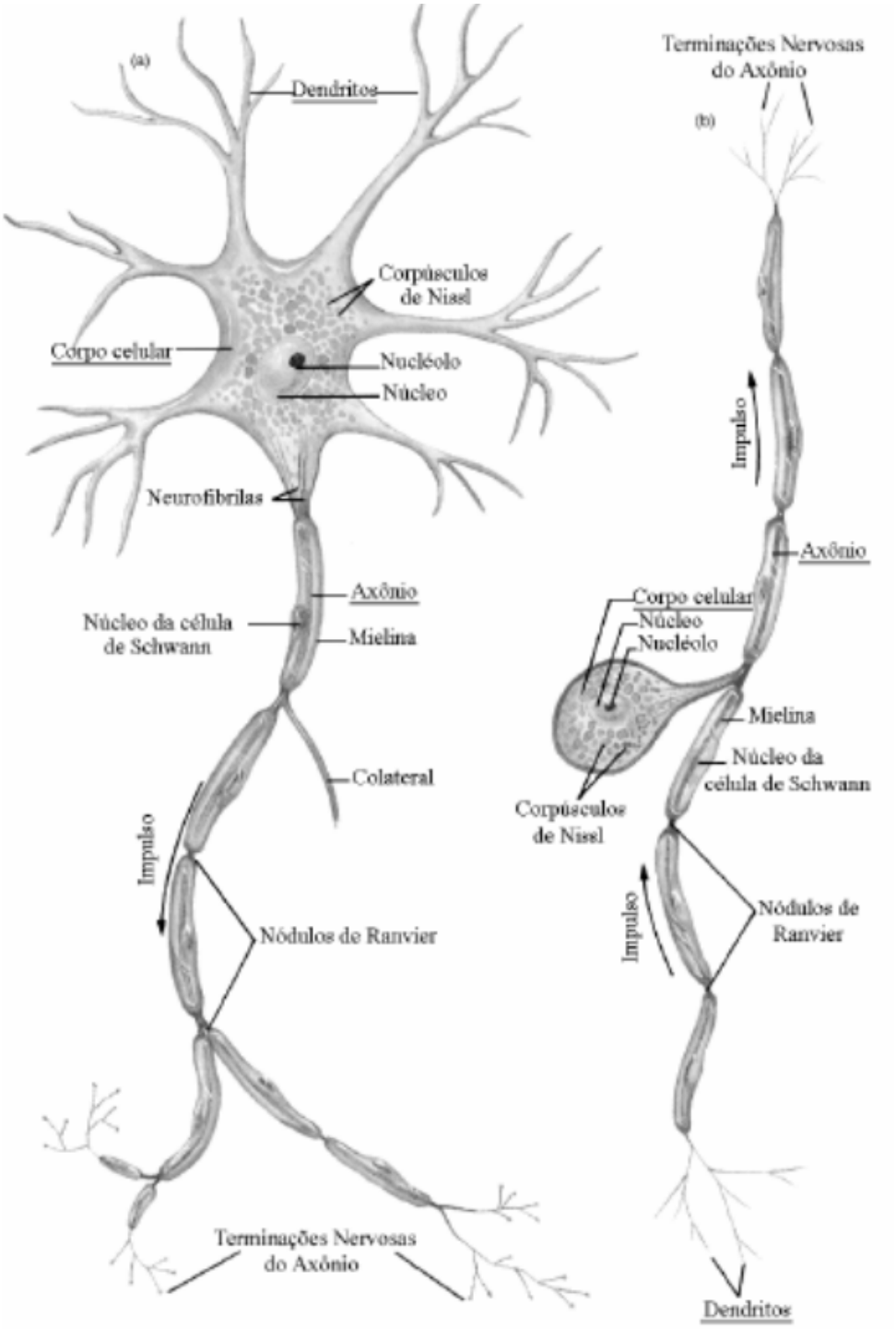

Figura 1: Representação dos neurônios motor (a) e sensorial (b) [1].

No geral, o corpo celular tem forma aproximadamente esférica ou piramidal. Os dendritos são extensões delicadas em forma de tubo que tendem a se ramificar e formar uma árvore espessa ao redor do corpo celular, o que permite uma grande área de contato para a recepção dos sinais de entrada.

O axônio se estende para longe do corpo celular e fornece o caminho pelo qual os sinais podem viajar do corpo celular por longas distâncias para outras partes do cérebro e do

sistema nervoso. O axônio difere dos dendritos tanto na estrutura quanto nas propriedades de sua membrana externa.

A maioria dos axônios é mais longa e mais fina do que dendritos (até um metro de comprimento) e exibe um padrão de ramificação diferente: enquanto os ramos dos dendritos tendem a se agrupar perto do corpo celular, os ramos dos axônios tendem a surgir na extremidade da fibra onde ele se comunica com outros neurônios ou glândulas, contendo estruturas denominadas botões sinápticos [2].

A membrana do neurônio, como a membrana externa de todas as células, tem cerca de cinco nanômetros de espessura e consiste em duas camadas de moléculas lipídicas dispostas com suas extremidades hidrofílicas apontando para a água tanto no interior quanto no exterior da célula e com suas extremidades hidrofóbicas apontando para longe da água, formando o interior da membrana [2].

Ao receberem um impulso nervoso, essas estruturas liberam substâncias químicas (chamadas neurotransmissores) responsáveis por transmitir sinais de um neurônio para outro [3].

A função primordial do axônio é transmitir informações na forma de pulsos regenerativos (sem atenuação) para diferentes partes do sistema nervoso e do organismo. Seu papel no sistema nervoso periférico envolve os aferentes conduzindo informações sensoriais para dentro do sistema neural e os eferentes destinando os comandos do sistema nervoso central aos efetores do corpo.

O potencial de ação se define por rápidas variações nos potenciais interno e externo da membrana celular nervosa, seu desenvolvimento ocorre da seguinte forma:

1. Círculo vicioso de *feedback* positivo abrindo os canais de sódio:

   Levando em conta que a membrana da fibra nervosa permaneça sem perturbação, não há potencial de ação atuando no nervo normal. No entanto, ocorrendo um evento capaz de elevar de -90 milivolts para o nível zero [3] o potencial da membrana, o próprio aumento de tensão implica na abertura de múltiplos canais de sódio regulados por ela. Isso possibilita o influxo rápido de íons sódio (Na^+), resultando em um maior aumento do potencial de membrana e, consequentemente, na abertura de mais canais regulados pelo nível de tensão e permitindo um fluxo mais intenso de Na^+ para o interior da fibra.

   Tal processo é denominado círculo vicioso de *feedback* positivo e, uma vez que ele seja suficientemente intenso, permanece até a ativação (abertura) de todos os canais

de sódio regulados pela tensão. Então, em outra fração de milissegundo, o aumento do potencial de membrana causa o fechamento dos canais de sódio e a abertura dos canais de potássio (K^+), e o potencial de ação se encerra.

2. Repolarização:

   Após total despolarização dessa área da célula, a membrana torna-se novamente impermeável ao Na^+, embora se mantenha permeável aos íons K^+. Em virtude da alta concentração de íons positivos no interior da célula nervosa, grandes quantidades de íons potássio voltam a se difundir para o meio externo, fazendo com que essa região no interior da célula nervosa volte a ser negativa.

   Nessa situação o neurônio torna-se novamente apto a transmitir um novo impulso nervoso. Na Figura 2 está representada a variação da tensão de uma membrana nervosa durante o potencial de ação no pulso nervoso.

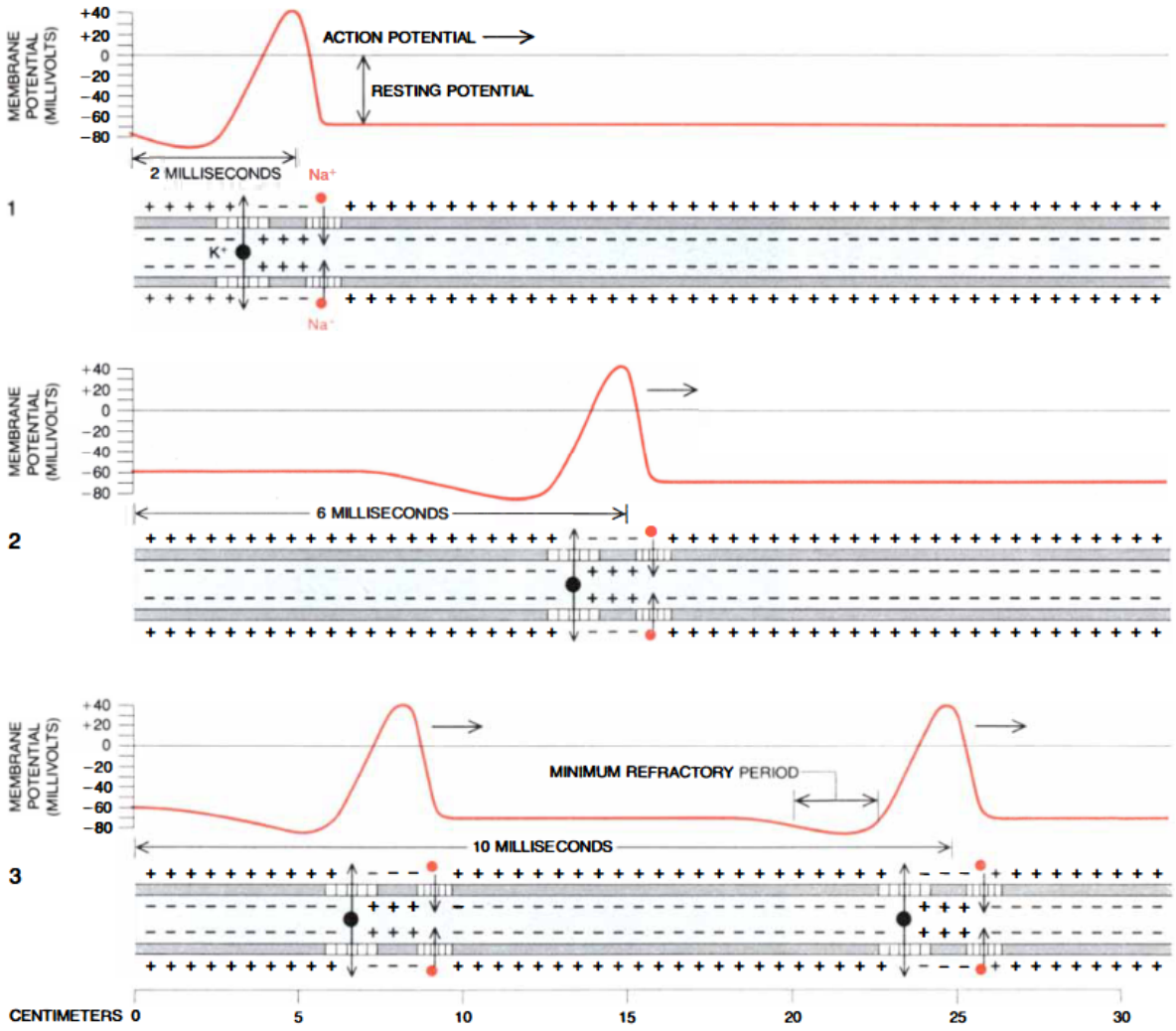

Figura 2: Propagação do impulso no nervo [2].

A descrição dada até então para a transmissão de um impulso nervoso se aplica a neurônios desprovidos de bainha de mielina. Nos mielinizados o potencial de ação ocorre apenas nos nódulos de Ranvier, pontos nos quais a membrana plasmática faz contato direto com o fluído intersticial. Nesse caso uma condução mais rápida e com menos gasto de energia, chama de saltatória.

A junção neuromuscular de grande fibra nervosa mielinizada com uma fibra muscular esquelética é representada na Figura 3. A fibra nervosa forma complexo de terminais ramificados que se invaginam na superfície extracelular da fibra muscular. Toda a estrutura é chamada de placa motora e é recoberta por uma ou mais células de Schwann que a isolam dos líquidos circunjacentes.

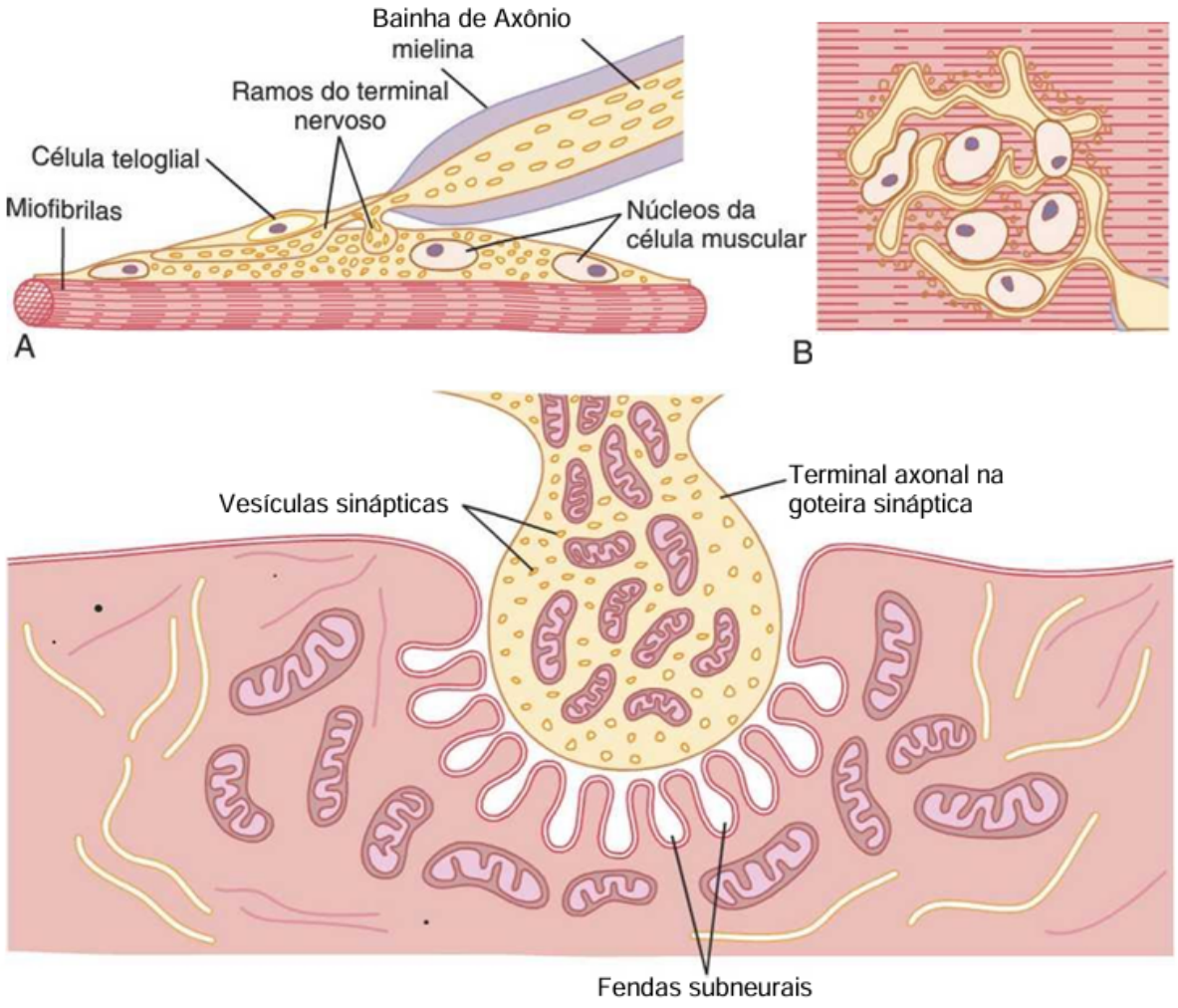

Figura 3: Diferentes perspectivas da placa motora: A - corte longitudinal através da placa motora; B - visão da superfície da placa motora; C - aspecto na micrografia eletrônica do ponto de contato entre um terminal isolado de um axônio e a membrana da fibra muscular. [3].

As membranas de ambas as células distam de uma fenda sináptica com entre 20 a 30 nm. Na membrana muscular, em face da goteira sináptica, existem dobras menores chamadas de fendas subneurais responsáveis por aumentar a área de superfície de atuação do transmissor sináptico.

Mediante a chegada de um potencial de ação no terminal do axônio, a acetilcolina é secretada na fenda sináptica. Essa liberação torna a membrana muscular mais permeável aos íons Na^+ devido à ação dos receptores de acetilcolina situados nas fendas subneurais.

O influxo abrupto desses íons no músculo resulta em um potencial de ação muscular que se propaga nas duas direções da fibra e se alastra da mesma maneira que nas membranas neurais.

O potencial de ação despolariza a membrana da fibra muscular e também penetra profundamente no interior dessa. Aproximadamente $0,2$ ms após ser liberada pelas vesículas sinápticas, a acetilcolina é metabolizada em ácido acético e colina pela enzima acetilcolinesterase.

Essa reação ocorre para que a membrana muscular, localizada na fenda sináptica, diminua a permeabilidade ao potássio deixando a placa motora novamente apta para receber um novo estímulo.

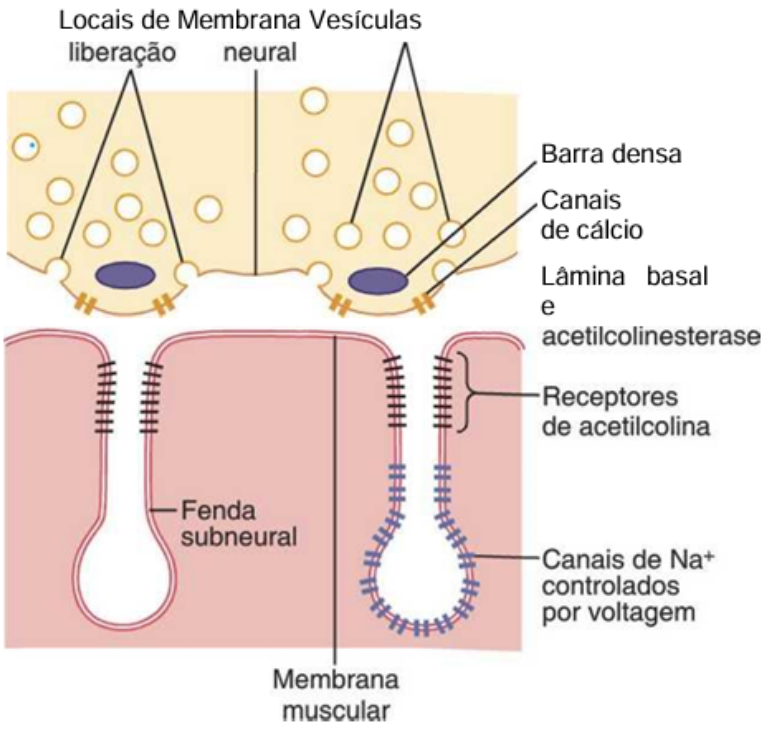

Figura 4: Liberação de acetilcolina das vesículas sinápticas na membrana neural da junção neuromuscular. [3].

1.1.1 Impulsos Elétricos pela Bainha de Mielina

A bainha de mielina desempenha um papel crucial na transmissão de impulsos elétricos, ou potenciais de ação, ao longo das fibras nervosas. Este processo é caracterizado principalmente pela condução saltatória, que aumenta significativamente a velocidade e a eficiência da propagação do sinal em axônios mielinizados.

Estrutura da Bainha de Mielina

A mielina é uma camada isolante rica em lipídios que envolve os axônios dos neurônios em segmentos, criando lacunas conhecidas como nós de Ranvier. Esses nós são essenciais para a condução de impulsos elétricos, pois contêm uma alta densidade de canais de sódio dependentes de voltagem.

Em contraste com as fibras não mielinizadas (nas quais os potenciais de ação se propagam continuamente ao longo de toda a membrana) as fibras mielinizadas permitem que os impulsos "saltem"de um nó para o outro, aumentando muito a velocidade de condução. [9]

Mecanismo de Condução Saltatória

- Despolarização nos nós - Quando um potencial de ação atinge um nó de Ranvier, ele causa a abertura de canais de sódio dependentes de voltagem. Íons de sódio (Na^+) correm para o axônio, levando à despolarização naquele ponto específico;

- Fluxo do circuito local - O influxo de íons de sódio gera um circuito local que despolariza a membrana adjacente no próximo nó. Como a mielina atua como um isolante, essa corrente não pode fluir através das seções mielinizadas (internós), mas, em vez disso, viaja através do axoplasma (o citoplasma dentro do axônio) para atingir o próximo nó [10];

- Transmissão rápida - A alta resistência e a baixa capacitância da bainha de mielina implicam que menos energia é necessária para despolarizar a membrana entre os nós. Como resultado, os potenciais de ação podem viajar muito mais rápido (até 100 metros por segundo) em comparação com 1-4 metros por segundo em fibras não mielinizadas [10].

A natureza segmentar da mielinização permite o uso eficiente de espaço e energia. Ao minimizar a área que deve ser despolarizada durante a transmissão do impulso, os axônios mielinizados podem manter uma comunicação rápida por longas distâncias sem precisar aumentar seu diâmetro significativamente [10]. Essa adaptação é particularmente

importante em vertebrados, permitindo funções complexas do sistema nervoso, apesar de tamanhos corporais maiores.

1.2 Fisiologia do Músculo Esquelético

A arquitetura do músculo esquelético é dada por um arranjo muito particular e bem descrito de fibras musculares (também chamadas de miofibras ou células musculares) e tecido conjuntivo associado [11]. As fibras musculares esqueléticas são de formato cilíndrico com diâmetros variando entre 10 e 100 μm [12], multinucleadas e pós-mitóticas.

Cada fibra é constituída por uma membrana plasmática chamada de sarcolema contendo centenas de miofibrilas, núcleos celulares e o retículo sarcoplasmático [1]. Na maior parte, cada núcleo dentro de uma fibra muscular controla o tipo de proteína sintetizada naquela região específica da célula. Essas regiões são conhecidas como domínios nucleares e têm um tamanho altamente regulado, mas não constante [13].

Conforme representado na Figura 5 as miofibrilas contém microfilamentos constituídos por miosina e actina. A disposição espacial dessas proteínas no tecido faz com que o músculo apresente aspecto estriado. No citoplasma, os íons de cálcio (Ca^{+}) formam um complexo com tais proteínas levando-as a deslizar uma em direção à outra, caracterizando a contração muscular. Uma vez cessado o estímulo, restabelece-se o sistema de transporte ativo do retículo, interrompendo a contração.

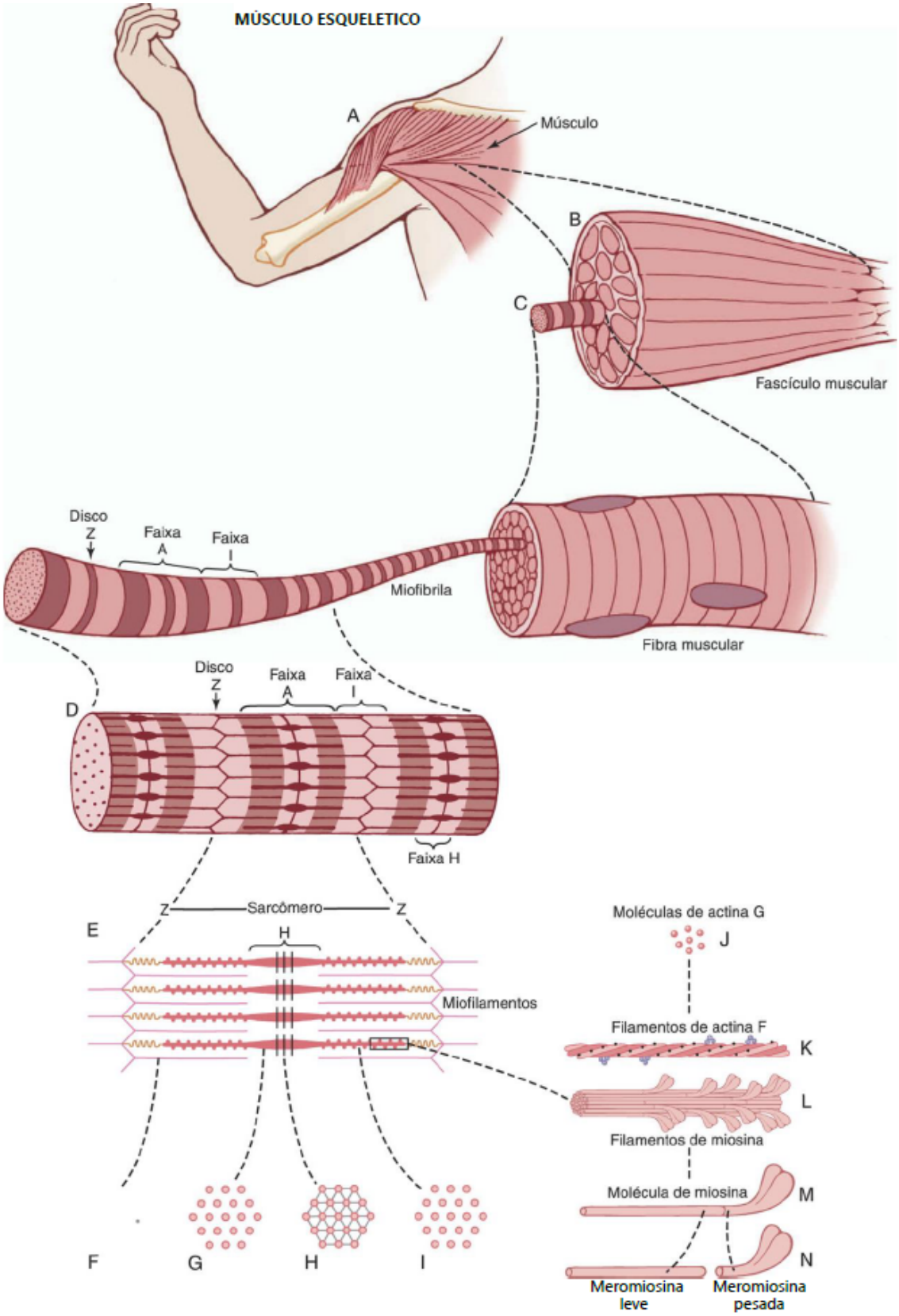

Figura 5: Organização do músculo esquelético do nível macroscópio ao molecular, onde as letras F,G,H e I indicam cortes transversais. [3].

A unidade motora é o termo utilizado para descrever a menor unidade muscular controlável. Uma unidade motora é constituída por um neurônio motor, suas junções neuro-

musculares e as fibras musculares inervadas por esse neurônio (Figura 6).

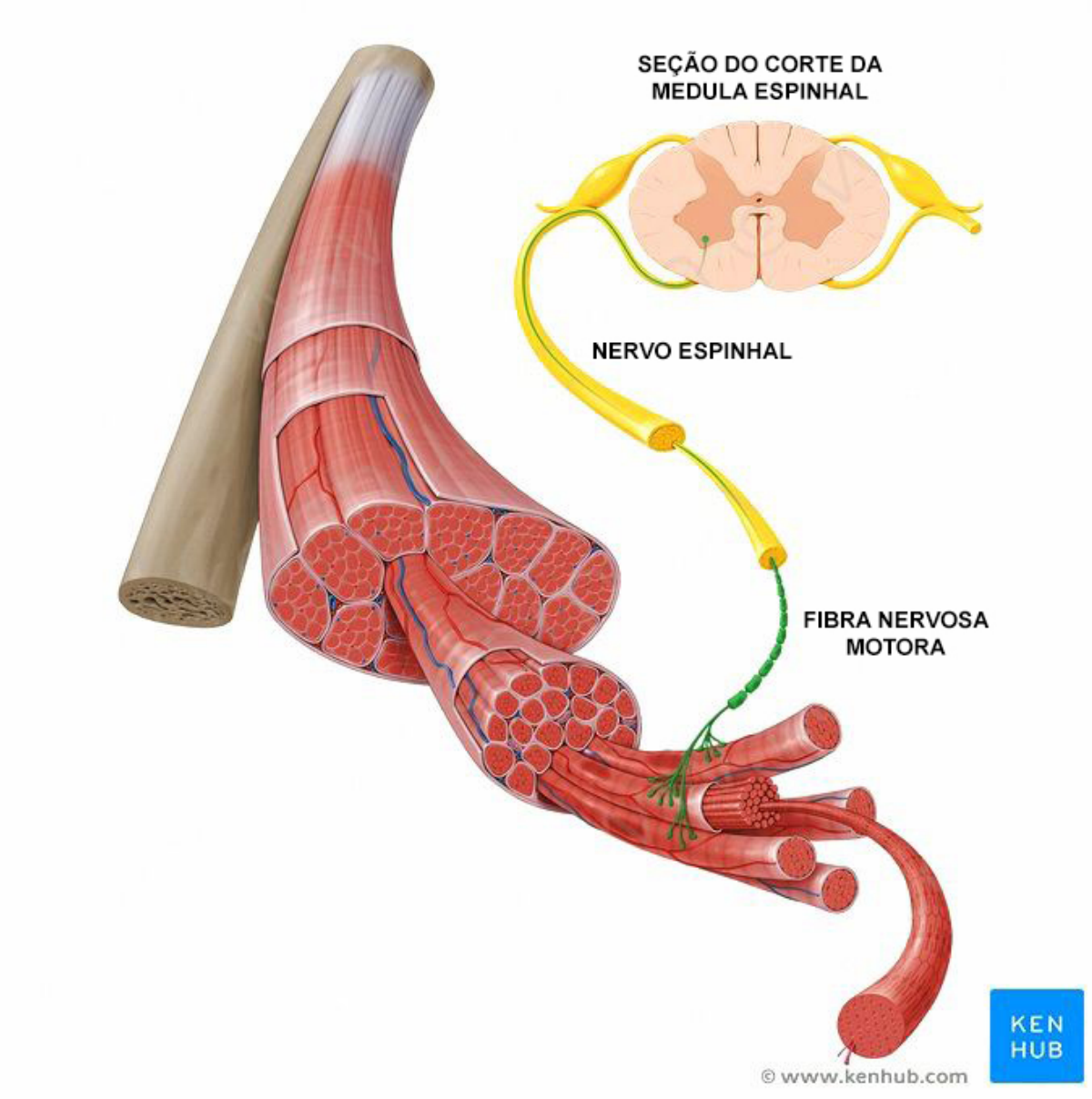

Figura 6: Neurônio motor. Adaptado de [4].

1.3 Anatomia e Função dos Músculos do Antebraço

Por apresentarem boa coordenação motora e propriocepção, os membros superiores são capazes de realizar com maestria atividades recreativas ou laborais, precisas ou grossas. As mãos são instrumentos delicados do indivíduo e representam o seu elo com o ambiente [14].

Para assegurar que isso ocorra da maneira correta, os músculos que ditam o movimento das mãos são essenciais:

1. Músculos intrínsecos - músculos que apresentam origem e inserção nela própria;

2. Músculos extrínsecos em relação à mão - músculos originados no antebraço e no cotovelo.

Existem vinte músculos no antebraço, divididos entre os compartimentos anterior (flexor) e posterior (extensor); cada compartimento recebe a subdivisão de superficial e profundo [15].

A parte superficial do compartimento anterior do antebraço apresenta um total de cinco músculos [15]:

- Pronador redondo;
- Flexor radial do carpo;
- Palmar longo;
- Flexor ulnar do carpo;
- Flexor superficial dos dedos.

E a parte profunda desse compartimento contém três [15]:

- Pronador quadrado;
- Flexor profundo dos dedos;
- Flexor longo do polegar.

A parte superficial do compartimento posterior do antebraço apresenta um total de sete músculos [15]:

- Braquiorradial;
- Extensor radial curto do carpo;
- Extensor radial longo do carpo;
- Extensor ulnar do carpo;
- Extensor dos dedos;
- Extensor do dedo mínimo;
- Ancôneo.

E a parte profunda desse compartimento contém cinco [15]:

- Músculo abdutor longo do polegar;
- Músculo extensor longo do polegar;
- Músculo extensor curto do polegar;
- Músculo extensor do indicador;
- Músculo supinador.

No movimento de preensão é necessária uma combinação entre o polegar e os outros dedos, sendo o anelar e o mínimo os mais importantes para a realização do movimento. O polegar é o responsável por envolver o objeto no plano contrário aos dedos, devido a isso ele é chamado de polegar opositor [16].

A mão necessita que ocorra um perfeito sinergismo entre a musculatura extrínseca/intrínseca e flexora/extensora para que haja um adequado movimento, seja ele de precisão ou de força [16].

Os músculos flexores extrínsecos dos dedos proporcionam a principal força para preensão. A sinergia, nesse caso, se dá com o extensor comum dos dedos fornecendo estabilidade aos flexores, os interósseos rodando a primeira falange e flexionando a articulação metacarpofalangeana e com os músculos tenares e adutor do polegar proporcionando força de compressão contra o objeto. Vale salientar que a posição de maior preensão palmar é a posição funcional do punho caracterizada pela amplitude entre 10° de flexão e 35° de extensão [17].

A preensão envolve um movimento que começa com a abertura da mão, seguido pelo envolvimento do objeto com os dedos e o polegar. O punho é estendido em um ângulo de 30°, enquanto ocorre a flexão das articulações metacarpofalangianas e interfalangianas proximal e distal dos dedos, além d; a abdução do polegar. Para que essa tarefa ocorra são necessários os músculos: extensor comum dos dedos, extensor próprio do indicador, extensor próprio do dedo mínimo, extensor radial longo e curto do carpo, lumbricais e abdutor curto do polegar [17] [18].

Tendo em vista o exposto, qualquer alteração no sistema musculoesquelético do antebraço irá alterar a função da mão em realizar atividades da vida diária, laborais e esportivas [16].

1.4 Formas de Captação e Características dos Sinais Eletromiográficos

A eletromiografia nos permite compreender comportamentos motores intencionais e automáticos. Após os MUAPTs (*Motor Unit Action Potential Train*) percorrerem as fibras musculares e gerarem um campo eletromagnético nas redondezas das fibras, um eletrodo (situado dentro desse campo) realiza a detecção e análise do eletromiograma (EMG), ou seja, do potencial elétrico produzido durante as contrações musculares.

Os EMGs podem ser detectados diretamente, através da inserção de eletrodos no tecido muscular, ou indiretamente, com eletrodos de superfície colocados em áreas da pele localizadas logo acima do tecido muscular. Os EMGs de superfície geralmente transmitem informações sobre a ativação muscular, como a intensidade da contração muscular, a manifestação mioelétrica da fadiga muscular e o recrutamento de unidades motoras [6].

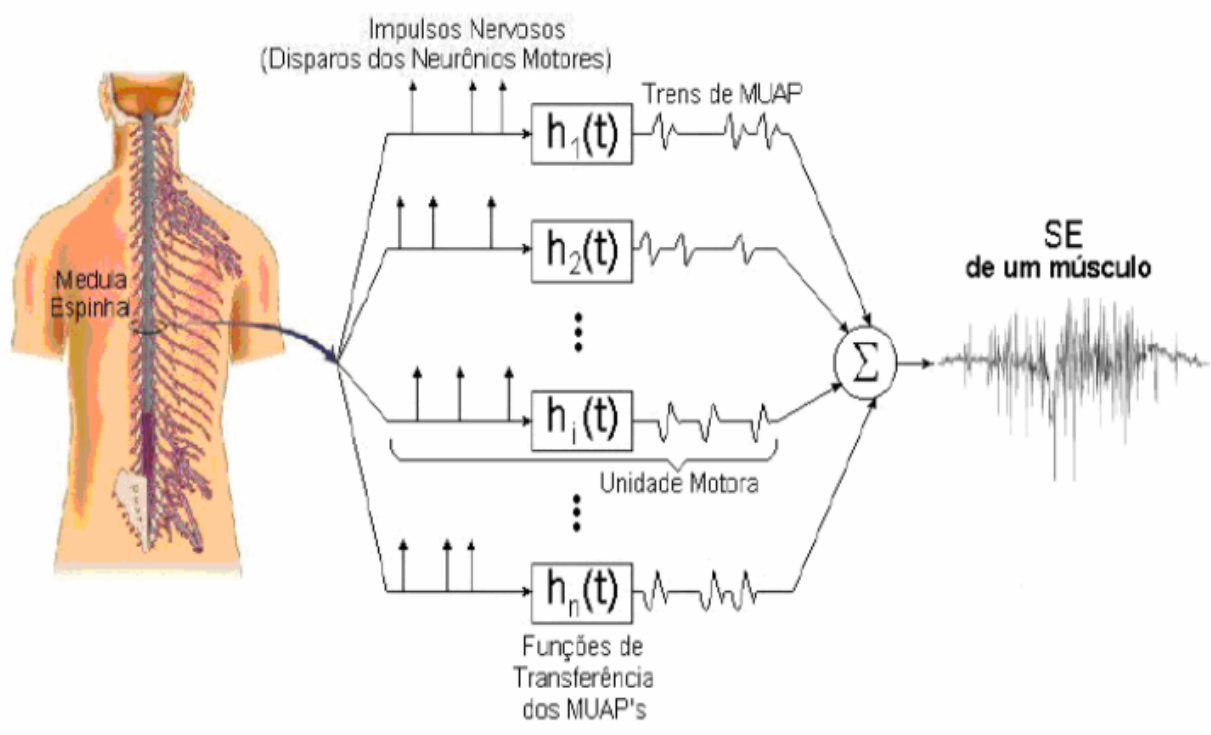

Figura 7: Representação esquemática da geração do sinal eletromiográfico de um músculo a partir da somatória dos trens de MUAPs das n unidades motoras desse tecido [5].

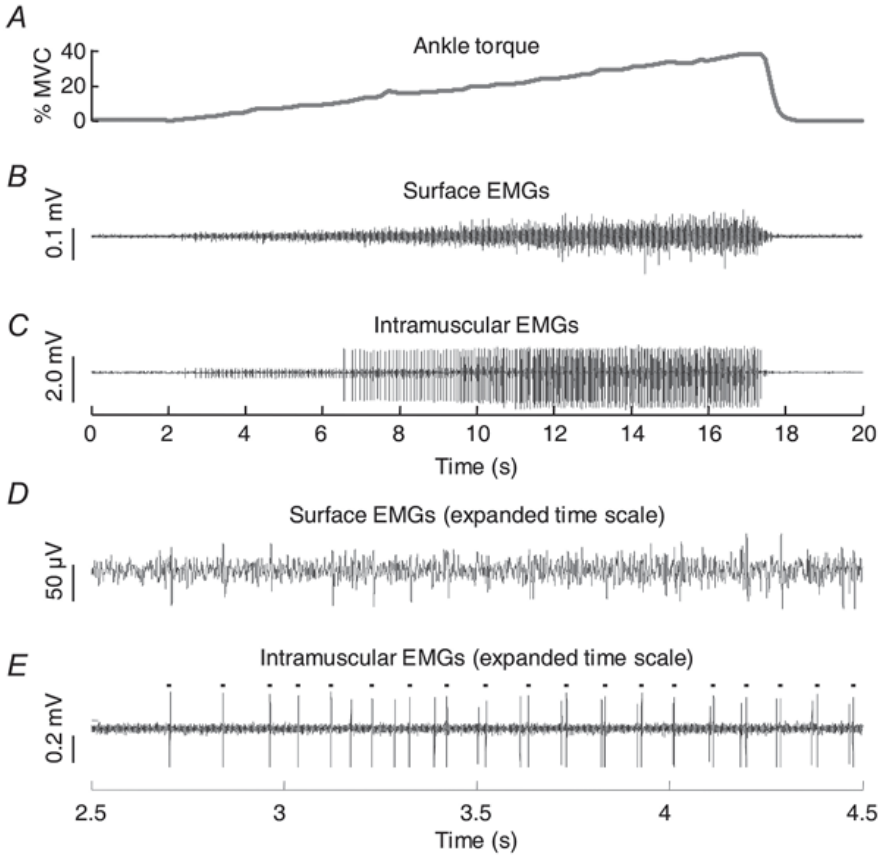

Figura 8: Eletromiogramas e potenciais de ação da unidade motora. A) mostra o torque de flexão plantar durante uma contração de rampa isométrica, de 0 a 40% MVC. EMGs de superfície e intramusculares registrados do músculo gastrocnêmio medial estão representados em B) e C), respectivamente. Breves epoches desses sinais são mostrados em D) e E). [6].

Levando em conta a diferença entre os MUAPs, as irregularidades na taxa de disparo dos neurônios motores e que em uma contração pode haver mais de um músculo envolvido, o sinal eletromiográfico foi descrito como estocástico [19] [20].

A limpeza da pele é útil para fornecer gravações de EMG com baixos níveis de ruído. A preparação adequada da pele garante a remoção de pelos corporais, óleos e camadas de pele escamosa e, consequentemente, reduz a impedância na interface eletrodo-gel-pele.

A atividade mioelétrica aparece na camada epitelial como potenciais elétricos com largura de banda limitada, de 15 a 400 Hz, e com amplitude muito pequena (de alguns micro a alguns mili-Volts) pico a pico (dependendo da intensidade da contração muscular) [6]. Instrumentos muito sensíveis são então necessários para a detecção, amplificação, condicionamento e digitalização de EMGs de superfície, de acordo com o diagrama de blocos simplificado mostrado na Figura 9.

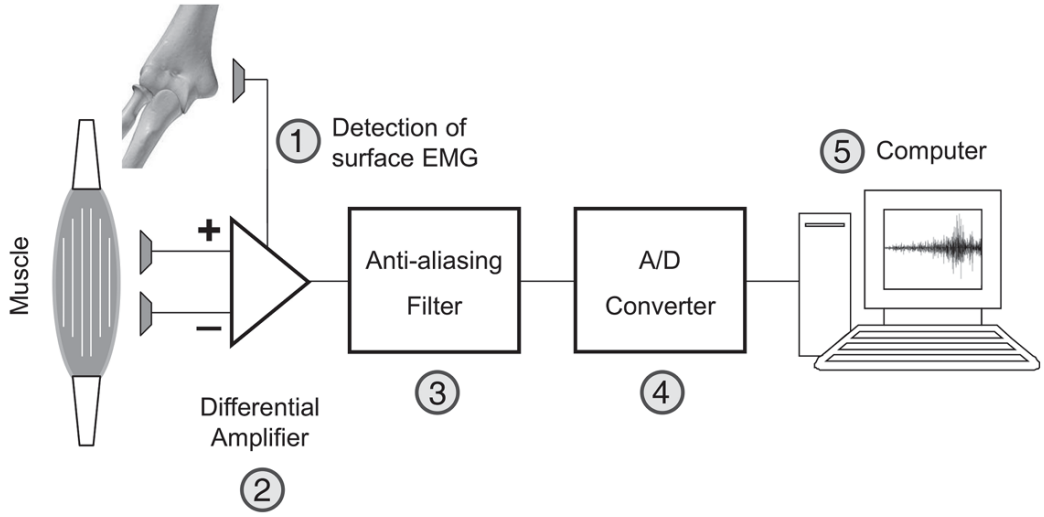

Figura 9: Diagrama de blocos simplificado mostrando cada uma das principais etapas referentes à aquisição de eletromiogramas de superfície: (1) a detecção dos potenciais mioelétricos com eletrodos de superfície e um eletrodo de referência, ilustrados esquematicamente no epicôndilo medial do úmero; (2) a amplificação desses potenciais com amplificadores diferenciais; (3) filtragem analógica dos potenciais amplificados para evitar aliasing e, finalmente; (4) a amostragem do eletromiograma de superfície em valores digitais de voltagem para serem armazenados em um computador (5) [6].

1.5 Impacto médico-social

A integração de tecnologias avançadas, como interfaces cérebro-computador e aprendizado de máquina, está revolucionando o campo das próteses. Por exemplo, empresas como a Ottobock estão utilizando IA para aprimorar suas próteses biônicas, permitindo que os usuários realizem movimentos complexos de forma mais natural. Essas inovações facilitam o controle das próteses e oferecem *feedback* sensorial, proporcionando uma experiência mais próxima da realidade.

O grande diferencial do desenvolvimento da inteligência artificial proposta, quando associado à reabilitação de pacientes paralisados e amputados, é a possibilidade que esses indivíduos controlem próteses ou órteses por meio de impulsos elétricos gerados pelo próprio cérebro, promovendo uma interação mais natural e intuitiva com as próteses.

Atualmente, a base de dados utilizada abrange apenas os dez movimentos mais simples, fazendo com que a IA seja limitada. No entanto, ao refiná-la e expandir seu treinamento para incluir uma gama mais ampla de movimentos e sinais eletromiográficos, será possível desenvolver próteses e órteses que mimetizam com precisão o funcionamento de membros

orgânicos. Essa evolução não apenas melhorará a funcionalidade das próteses, mas também contribuirá para a qualidade de vida dos usuários, permitindo-lhes realizar atividades cotidianas com maior facilidade e autonomia.

2 Introdução à Base de Dados

Esta monografia se fundamenta em uma base de dados composta por sinais EMG, que retratam 10 movimentos distintos da mão realizados por um grupo de participantes de diferentes perfis, visando explorar as respostas musculares do antebraço para cada movimento específico. Esses dados foram registrados utilizando o sistema BIOPAC MP36, equipado com 4 eletrodos bipolares de superfície do tipo Ag/AgCl, um dispositivo amplamente empregado em estudos de biomecânica e fisiologia.

Para a coleta foram chamados 40 participantes de características distintas para garantir que a IA desenvolvida seja capaz de identificar os movimentos de qualquer individuo. Ainda, cada participante realizou cinco repetições de cada movimento, totalizando um volume considerável de dados por participante, permitindo uma análise estatística robusta e a construção de modelos classificatórios com maior acurácia. Os sinais foram processados e filtrados em etapas posteriores para assegurar a remoção de ruídos e artefatos, melhorando a qualidade dos dados antes de sua aplicação nos algoritmos de aprendizado de máquina.

A respeito dos movimentos, esses englobam desde uma posição neutra até movimentos complexos de flexão e rotação, foram selecionados por sua relevância prática para aplicações em sistemas de controle baseados em EMG, como próteses e dispositivos de assistência. Os movimentos, identificados pelos números de 0 a 9, são descritos a seguir.

O primeiro movimento, a posição neutra, corresponde ao estado inicial, no qual o pulso e os dedos permanecem relaxados e sem movimentos significativos, como na Figura 10. Em seguida, têm-se a extensão do pulso, que se caracteriza pelo levantamento da mão, ativando predominantemente os músculos extensores do antebraço, Figura 11. A flexão do pulso é o movimento oposto à extensão, no qual a mão é movida para baixo, acionando os músculos flexores, Figura 12.

Figura 10: Antebraço na posição neutra (0)

Figura 11: Antebraço com pulso estendido (1)

Figura 12: Antebraço com pulso flexionado (2)

Além desses, o movimento de rotação ulnar refere-se ao desvio do pulso em direção ao lado ulnar, ou seja, ao lado do dedo mínimo, Figura 13. Já a rotação radial se dá pelo desvio do pulso para o lado radial, próximo ao polegar, Figura 14. O fechamento da mão em um punho envolve uma contração intensa de múltiplos grupos musculares do antebraço, Figura 15.

Figura 13: Antebraço com desvio ulnar (3)

Figura 14: Antebraço com desvio radial (4)

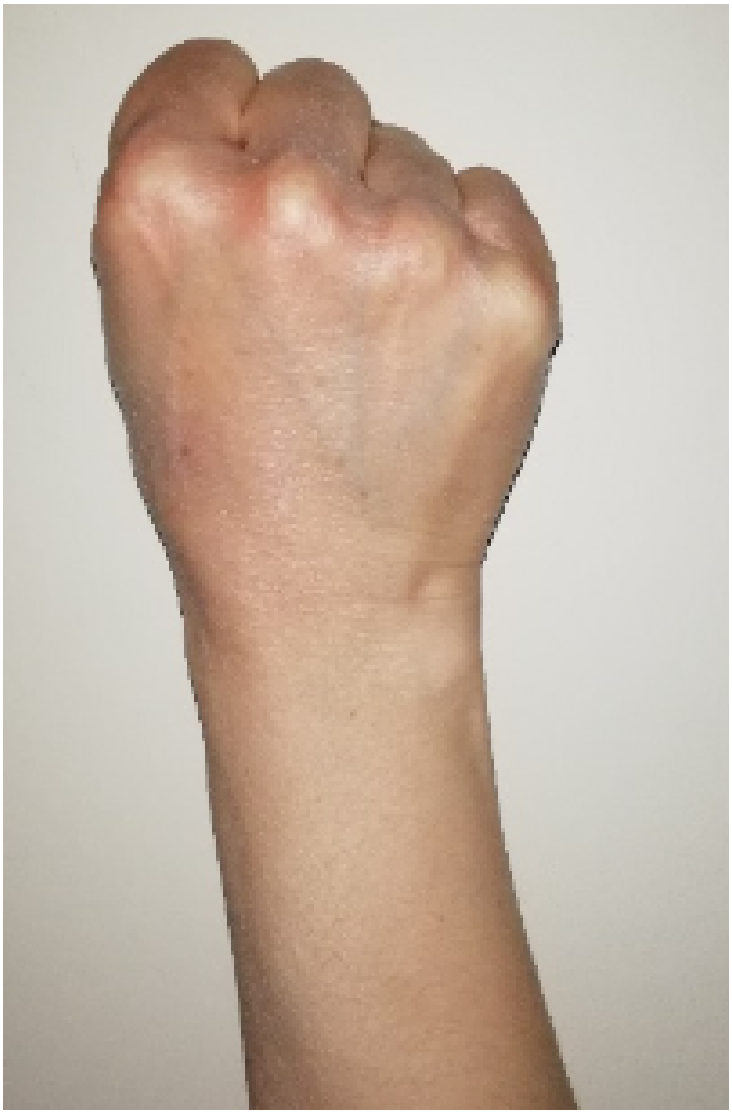

Figura 15: Antebraço com punho fechado (5)

Na sequência, a abdução dos dedos implica na abertura dos mesmos para fora, ativando músculos intrínsecos da mão e antebraço, Figura 16, enquanto a adução dos dedos representa o movimento inverso, em que os dedos se aproximam, Figura 17. Finalmente, os movimentos de supinação e pronação envolvem, respectivamente, a rotação do antebraço para que a palma da mão fique voltada para cima e para baixo, representados pelas Figuras 18 e 19. Vale ressaltar que esses 4 últimos movimentos são contínuos, ou seja, diferente dos anteriores, eles são registrados com uma sequência de dados discretos e essa representa o movimento como um todo, as imagens abaixo foram tiradas durante o movimento.

Figura 16: Antebraço com abdução dos dedos (6)

Figura 17: Antebraço com adução dos dedos (7)

Figura 18: Antebraço na posição supinada (8)

Figura 19: Antebraço na posição pronada (9)

Dessa forma, essa seleção de movimentos permite avaliar um conjunto amplo e diversificado de respostas musculares, possibilitando uma análise robusta dos sinais EMG e a sua aplicação em algoritmos de *machine learning* para classificação de movimentos.

3 IAs e suas Diferentes Estruturas

3.1 Funcionamento de uma IA

O princípio básico de operação de uma IA consiste na capacidade das máquinas simularem processos da inteligência humana por meio do aprendizado e refinamento mediante exposição a uma grande quantidade de dados. Tal processo se dá pela identificação de padrões e relações.

O aprendizado engloba a utilização de algoritmos, definidos pelo conjunto de instruções e regras que norteiam as análises e tomadas de decisão da IA. Exemplificando com um

subconjunto conhecido de IA, em machine learning os algoritmos são treinados em dados rotulados ou não rotulados, visando fazer previsões ou mesmo categorizar informações.

Outra aplicação é o deep learning, que se assemelhando à estrutura e à função do cérebro humano, faz uso de redes neurais artificiais com diversas camadas para processar as informações.

Assim, aliando apreensão de informações e adaptação constante, os sistemas de IA se refinam para tarefas específicas como reconhecimento de imagens, categorização de dados e outras.

3.2 Tipos de IA

3.2.1 Inteligência Artificial Estreita (ANI)

O termo "estreita" da nomenclatura refere-se à limitação da IA de armazenar uma grande quantidade de dados e realizar tarefas complexas, mas sempre visando o objetivo específico para o qual foram programadas.

A ANI não possui uma compreensão intrínseca dos dados que processa e carece de consciência própria. Em outras palavras, essa tecnologia precisa de vastas quantidades de dados tanto para o treinamento inicial quanto para o aprendizado contínuo ao longo do tempo.

Além disso, a ANI desempenha um papel fundamental no avanço tecnológico, especialmente em áreas como automação de processos, análise de dados e execução de tarefas específicas. No entanto, essa tecnologia permanece dependente da intervenção humana, tanto em sua concepção quanto em sua manutenção. Isso faz com que a Inteligência Artificial Estreita também seja popularmente chamada de "IA limitada" ou "IA fraca".

Incluídas nessa classificação, há duas subcategorias:

- Máquinas reativas: são projetadas com o intuito de responder a situações específicas com base nas informações que recebem no momento, sem guardar experiências anteriores ou aprender com elas. Isso significa que não armazenam muitos dados e reagem a apenas alguns estímulos de acordo com a maneira como foram configuradas, não tendo assim capacidade de influenciar decisões futuras;

- Memória limitada: é um tipo de sistema que pode utilizar dados passados para melhorar suas respostas e tomar decisões mais precisas no futuro. Diferentemente da IA de memória limitada, as máquinas reativas são capazes de armazenar tem-

porariamente informações de eventos recentes e usá-las para ajustar suas ações e decisões.

3.2.2 Artificial General Intelligence (AGI)

É esperado que a Inteligência Artificial Geral apresente o mesmo nível de habilidades cognitivas de um ser humano (explicando a sua classificação como "IA Forte").

A teoria enuncia que a AGI será capaz de repetir comportamentos como criatividade, percepção e aprendizado. Podendo resolver problemas, fazer previsões e transferir conhecimentos de uma área para outra.

3.2.3 Artificial Super Intelligence (ASI)

A Superinteligência Artificial é uma projeção futura do tipo mais avançado de IA, também fazendo parte do grupo denominado "IA Forte".

Estima-se que uma máquina portadora de ASI seja autoconsciente, com o poder de superar a capacidade e a inteligência humana em praticamente qualquer área.

3.3 Etapas de Funcionamento

Com o surgimento de vários canais de fontes de dados a pesquisa no campo da IA levou à definição de uma arquitetura de IA canônica que garante um ecossistema de ponta a ponta [21]. No entanto, vários estudos provaram que a arquitetura de IA requer algumas etapas para garantir que o processo seja mais eficiente e razoável em tempo de resposta [22]. Pode-se resumir essas etapas como:

1. Coleta e Preparação de Dados - Etapa que garante a prospecção de dados de múltiplos canais em diferentes formatos (dados estruturados e não estruturados);

2. Escolha do Modelo - É selecionado, mediante análise da tarefa almejada, o tipo de modelo apropriado;

3. Condicionamento de Dados - Etapa na qual dados heterogêneos (estruturados e não estruturados) são convertidos em dados de informação bruta após curadoria, padronização, gerenciamento e rotulagem de dados;

4. Treinamento do Modelo - O modelo ajusta seus parâmetros após alimentação com os dados de treinamento tendo como base uma função de perda que mede a diferença

entre as previsões feitas pelo modelo e as saídas reais. O intuito dessa etapa é transformar informação em conhecimento;

5. Validação e Ajuste - Etapa que visa avaliar o desempenho do modelo em dados que não foram utilizados durante o treinamento. Ela prima por evitar a ocorrência de problemas como o overfitting, onde o modelo se adapta excessivamente aos dados de treinamento, perdendo a capacidade de generalizar para novos exemplos;

6. Avaliação - Essa etapa verifica se o modelo aprendeu a partir dos dados e se é capaz de fazer previsões precisas em dados não vistos. Essa avaliação verifica problemas como:

    - Overfitting - Quando há um bom ajuste aos dados de treinamento mas não é capaz de generalizar para novos dados;
    - Underfitting - Quando o modelo é insuficiente na tarefa de capturar os padrões subjacentes nos dados, resultando em um desempenho fraco tanto nos dados de treinamento quanto nos dados de teste.

7. Trabalho em equipe homem-máquina - Etapa na qual ocorre a colaboração de humanos para converter o "conhecimento" em "insight", que orienta a execução das ações ou decisões subsequentes [22].

3.4 Modelos de Treinamento de Inteligência Artificial

Sabendo que o treinamento de modelos de inteligência artificial é uma etapa fundamental para capacitar algoritmos a executarem tarefas específicas, nessa seção são abordados os principais tipos de aprendizado em IA e suas características com base na literatura científica.

3.4.1 Aprendizado Supervisionado

Aprendizado supervisionado é a tarefa de machine learning na qual o algoritmo aprende uma função que mapeia uma entrada para uma saída com base em pares de entrada-saída de exemplo [7].

No aprendizado supervisionado, os algoritmos necessitam de assistência externa. O conjunto de dados de entrada é dividido em conjunto de dados de treinamento e teste,

enquanto que o conjunto de dados de treinamento tem variável de saída que precisa ser prevista ou classificada.

Cada exemplo é descrito por um vetor de valores (atributos) e pelo rótulo da classe associada. O objetivo do algoritmo é construir um classificador que possa determinar corretamente a classe de novos exemplos ainda não rotulados [23]. Esse método de aprendizado é o mais utilizado. Todos os algoritmos aprendem algum tipo de padrão do conjunto de dados de treinamento e os aplicam ao conjunto de dados de teste para previsão ou classificação. O fluxo de trabalho desse tipo de treinamento é fornecido na Figura 20.

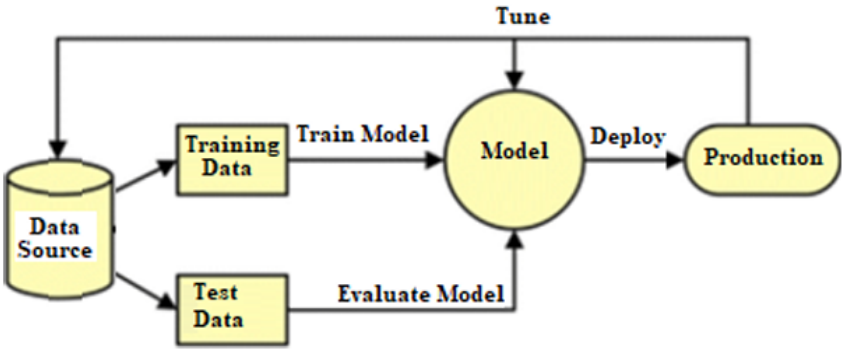

Figura 20: Fluxo de trabalho de aprendizagem supervisionada [7].

Algoritmos Comuns

1. Classificação - Algoritmos como regressão logística, máquinas de vetor suporte (SVM), árvores de decisão e redes neurais usados para classificar dados em categorias discretas;

2. Regressão - Métodos como regressão linear e regressão polinomial são utilizados para prever valores contínuos com base em variáveis independentes.

3.4.2 Aprendizado Não Supervisionado

O aprendizado não supervisionado é aquele no qual o modelo é treinado sem a presença de rótulos ou respostas corretas previamente definidas. O principal objetivo é identificar padrões ocultos pela exploração dos dados, agrupando instâncias semelhantes ou reduzindo a dimensionalidade dos dados para facilitar a análise.

Tal abordagem é especialmente útil quando não se tem conhecimento prévio sobre as características dos dados ou quando os dados rotulados são escassos ou inexistentes. O fluxo de trabalho desse tipo de treinamento é fornecido na Figura 21.

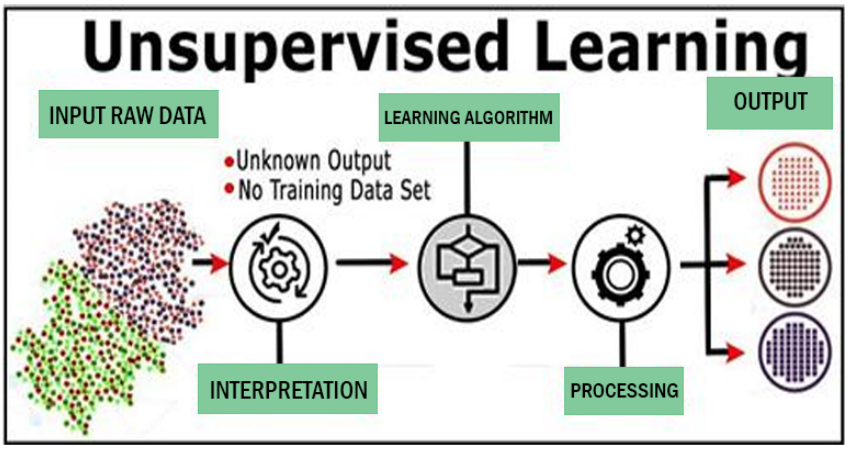

Figura 21: Fluxo de trabalho de aprendizagem não supervisionada [7].

Algoritmo Comum Empregado no Código

- **K-Means Clustering**

K-means é um dos algoritmos de aprendizado não supervisionado mais simples que resolve o conhecido problema de clustering (agrupamento) [7].

Ele busca dividir um conjunto de dados em k clusters distintos. Cada cluster é representado por um centro, que é o ponto médio (ou centróide) de todos os pontos de dados que pertencem a esse cluster.

O objetivo principal desse algoritmo é minimizar a variância dentro de cada cluster agrupando os dados de forma que os pontos dentro do mesmo cluster sejam o mais semelhantes possível entre si.

Procedimento do K-means

1. Definição de k - O usuário determina o número de clusters que deseja formar;

2. Inicialização dos Centros - Os centros devem ser inicialmente escolhidos de maneira estratégica, uma vez que a determinação das posições iniciais dos centros é crucial para o sucesso do algoritmo. Uma boa prática é colocar os centros o mais longe possível uns dos outros para garantir que eles cubram bem a diversidade dos dados;

3. Atribuição de Clusters - Fazendo uso da distância Euclidiana, cada ponto de dado é atribuído ao cluster cujo centro está mais próximo;

4. Atualização dos Centros - Após todos os pontos terem sido atribuídos a um cluster, os centros dos clusters são recalculados como a média dos pontos atribuídos a cada um;

5. As etapas de atribuição e atualização são repetidas até que não haja mais mudanças significativas nas atribuições dos clusters ou até que um número máximo de iterações seja alcançado.

O K-means é representado pela Figura 22.

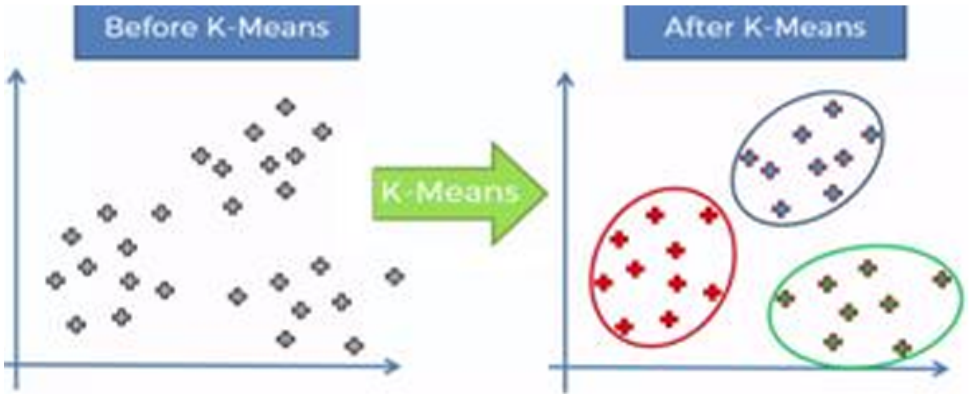

Figura 22: Fluxo de trabalho de aprendizagem não supervisionada [7].

3.5 Estruturas e Bibliotecas Empregadas

3.5.1 TensorFlow

TensorFlow é um framework (estrutura) de código aberto desenvolvido pelo Google que permite a criação de modelos de machine learning e deep learning através de uma interface que facilita a construção de gráficos computacionais. Tais gráficos retratam operações matemáticas com conexões que representam tensores (estruturas de dados multidimensionais).

O TensorFlow é otimizado para ser executado em diferentes plataformas, incluindo CPUs, GPUs e TPUs (Tensor Processing Units).

3.5.2 Keras

Keras é uma biblioteca de alto nível para desenvolvimento de redes neurais em Python, projetada para facilitar a prototipagem rápida e intuitiva de modelos de deep learning. Ela é construída sobre frameworks (como TensorFlow), permitindo que desenvolvedores criem, compilem e treinem redes neurais com facilidade.

Keras suporta diversas arquiteturas de redes neurais, como redes convolucionais e recorrentes, sendo amplamente utilizada em aplicações de reconhecimento de imagem e processamento de linguagem natural.

3.5.3 NumPy

NumPy (Numerical Python) é uma biblioteca que fornece estruturas de dados eficientes, como arrays multidimensionais (ndarrays), além de funções matemáticas que permitem operações vetorizadas.

O NumPy é amplamente utilizado em ciência de dados e machine learning por ser eficiente em manipular grandes conjuntos de dados e realizar cálculos complexos de forma rápida.

3.5.4 Pandas

Pandas é uma biblioteca poderosa para manipulação e análise de dados em Python que fornece estruturas como DataFrames e Series que facilitam o trabalho com dados rotulados ou relacionais.

Essa biblioteca é amplamente utilizada em tarefas como limpeza de dados, análise exploratória e manipulação eficiente de grandes conjuntos de dados. Sua integração com outras bibliotecas populares torna-a uma ferramenta muito versátil.

3.5.5 Matplotlib

Matplotlib é uma biblioteca popular para visualização de dados em Python, permitindo a criação de gráficos estáticos, animados e interativos (incluindo histogramas, gráficos de dispersão e gráficos 3D) com facilidade. Ela é frequentemente utilizada em conjunto com outras bibliotecas como NumPy e Pandas para representar visualmente dados analisados.

3.5.6 Scikit-learn

Scikit-learn é uma biblioteca robusta para aprendizado de máquina em Python a qual oferece uma ampla gama de algoritmos para tarefas como classificação, regressão e agrupamento, além de ferramentas para pré-processamento e avaliação de modelos.

Projetada para ser de fácil utilização, integra-se bem com outras bibliotecas como NumPy e Pandas.

3.5.7 Seaborn

Seaborn é uma biblioteca baseada no Matplotlib que fornece uma interface mais amigável para a criação de gráficos estatísticos. Ela facilita a visualização dos dados ao oferecer funções específicas que representam relações estatísticas entre variáveis.

Seaborn é especialmente útil para explorar conjuntos de dados complexos e gerar visualizações informativas com menos código.

3.5.8 Graphviz

Graphviz é uma ferramenta para visualização gráfica que permite criar diagramas a partir da descrição textual dos gráficos. É frequentemente utilizada para representar estruturas hierárquicas ou fluxos complexos em algoritmos, incluindo os usados em machine learning.

Com Graphviz, os usuários podem gerar visualizações claras e compreensíveis dos modelos ou processos que estão analisando.

3.5.9 OS

A biblioteca OS (Operating System) do Python fornece uma maneira conveniente de interagir com o sistema operacional dando a possibilidade de realizar operações como manipulação de arquivos e diretórios, execução de comandos do sistema operacional e acesso a informações do ambiente do sistema.

A biblioteca OS é essencial para scripts que precisam interagir com o sistema subjacente ou gerenciar arquivos durante o processamento de dados.

4 Análise dos Dados

4.1 K-means e o Método do Cotovelo

O método do cotovelo é uma técnica amplamente utilizada para determinar o número ideal de *clusters* em algoritmos de agrupamento, especialmente no K-means (abordado na Seção 3.4.2). Este método é fundamental no contexto de aprendizado não supervisionado, onde a identificação do número adequado de *clusters* pode impactar significativamente a qualidade da segmentação dos dados.

O método do cotovelo busca identificar o ponto onde a adição de mais *clusters* resulta em um ganho marginal na explicação da variância dos dados.

1. O método do cotovelo é expresso pela Soma do Erro Quadrado [24]:

$$SSE = \sum_{k=1}^{k} \sum_{x_i \in S_k} ||X_i - C_k||_2^2 \tag{4.1}$$

Levando em conta que k = número de *clusters* formados, C_i = i-ésimo *cluster* e x = os dados presentes em cada *cluster*.

2. A determinação inicial do centroide é feita aleatoriamente a partir dos objetos disponíveis até o *cluster* k. Para o calculo do próximo centroide do i-*cluster*, a seguinte fórmula é empregada:

$$v = \frac{\sum_{i=1}^{n} x_i}{n} \; onde \; i = 1,\ 2,\ 3,\ \ldots,\ n \tag{4.2}$$

3. A distância de cada objeto a cada centroide é calculada usando a Distância Euclidiana:

$$d(x, y) = ||x - y|| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \; onde \; i = 1,\ 2,\ 3,\ \ldots,\ n \tag{4.3}$$

Levando em conta que x_i = x em função de i, y_i = variável de saída e n = número de objetos.

4. Cada objeto é alocado no centroide mais próximo.

5. Cada objeto é alocado em um *cluster* na iteração com k-means considerando a distância ao ponto central do *cluster*.

6. Após a iteração e processamento, o novo centróide é calculado pela Equação 4.2.

7. A etapa 3 deve ser repetida caso a nova posição do centróide com o antigo não for a mesma.

Esse método afirma que quanto menor a inércia, melhor a divisão dos *clusters* e o número ideal é determinado pelo ponto de cotovelo, caracterizado por onde a curva começa a se inclinar mais suavemente.

Este é o ponto onde a inércia começa a diminuir a uma taxa menor, indicando que adicionar novos *clusters* não resulta em uma melhora significativa na compactação com os anteriores. O número de *clusters* ideal é aquele que está no ponto de cotovelo, pois representa um bom *trade-off* entre minimizar a inércia e não adicionar *clusters* desnecessários.

4.1.1 Código

```
#cluster  C digo com K-means

import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data

def plot_elbow_method(data, max_k=10):
    inertias = []
    K = range(1, max_k + 1)

    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        inertias.append(kmeans.inertia_)

    plt.figure(figsize=(8, 4))
    plt.plot(K, inertias, 'bo-')
    plt.xlabel('N mero de Clusters')
    plt.ylabel('In rcia')
    plt.title('M todo do Cotovelo para Encontrar o N mero Ideal de
    Clusters')
    plt.show()

# Especificar o caminho para o arquivo no Google Drive
data_path = "/content/drive/My Drive/random_output.csv"

# Verificar se o caminho existe
if not os.path.exists(data_path):
    raise FileNotFoundError(f"Data path {data_path} does not exist.")
```

```
39
40 # Carregar e normalizar os dados
41 data = load_and_normalize_data(data_path)
42
43 # Aplicar o M todo do Cotovelo para encontrar o n mero ideal de clusters
44 plot_elbow_method(data, max_k=10)
```

Tabela 1: Código em *cluster* k-means.

4.1.2 Estrutura do Código

Importação de Bibliotecas

```
1 import os
2 import numpy as np
3 import pandas as pd
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.cluster import KMeans
6 import matplotlib.pyplot as plt
```

Tabela 2: Importação de Bibliotecas (K-means e o Método do Cotovelo).

- OS: Introduzida para a interação com o sistema operacional;
- NumPy: Anexada ao código para a manipulação de *arrays*;
- Pandas: Usada para manipulação e análise de dados, especialmente para ler arquivos CSV;
- StandardScaler: Uma classe do sklearn que normaliza os dados, removendo a média e escalando para a variância unitária;
- K-means: Implementação do algoritmo K-means para *clustering*;
- Matplotlib.pyplot: Destinada para criação de gráficos e visualizações.

Função *load_and_normalize_data*

```
1 def load_and_normalize_data(file_path):
2     df = pd.read_csv(file_path)
3     data = df.iloc[:, :-1].values.astype(np.float32)
```

```
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data
```

Tabela 3: Função *load_and_normalize_data* (K-means e o Método do Cotovelo).

As etapas de funcionamento da função em questão são descritas abaixo:

1. Carregamento dos Dados: Um arquivo CSV é lido usando $pd.read\_csv()$, armazenando os dados em um DataFrame df;

2. Seleção de Colunas: $df.iloc[:, :-1]$ seleciona todas as colunas, com exceção da última. A última coluna geralmente contém rótulos ou categorias que não são usadas no *clustering*;

3. Conversão de Tipo: Converte os dados para o tipo $float32$ para economizar memória e melhorar a performance;

4. Normalização: Cria uma instância de StandardScaler, ajusta e transforma os dados. A normalização é importante para que todas as características tenham a mesma escala, evitando que características com maiores magnitudes dominem o cálculo das distâncias no K-means;

5. A função retorna os dados normalizados.

Função *plot_elbow_method*

```
def plot_elbow_method(data, max_k=10):
    inertias = []
    K = range(1, max_k + 1)

    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        inertias.append(kmeans.inertia_)

    plt.figure(figsize=(8, 4))
    plt.plot(K, inertias, 'bo-')
    plt.xlabel('N mero de Clusters')
    plt.ylabel('In rcia')
```

```
14     plt.title('M todo do Cotovelo para Encontrar o N mero Ideal de
       Clusters')
15     plt.show()
```

Tabela 4: Função *plot_elbow_method* (K-means e o Método do Cotovelo).

As etapas de funcionamento da função em questão são descritas abaixo:

1. Inicialização da Lista de Inércia: Cria uma lista vazia para armazenar a inércia (ou soma das distâncias quadradas das amostras aos seus centros de cluster) para cada valor de k;

2. Definição do Intervalo de k: $K = range(1,\ max\_k + 1)$ define o intervalo de valores de k (número de *clusters*) que será testado;

3. Cálculo da Inércia para cada valor de k em K: Cria uma instância do K-means com o número atual de *clusters*; Ajusta o modelo aos dados usando $kmeans.fit(data)$; Adiciona a inércia calculada ($kmeans.inertia\_$) à lista "*inertias*";

4. Plotagem do Gráfico: Cria uma figura com tamanho específico, plota os valores de k no eixo x e as inércias no eixo y, adiciona rótulos e título ao gráfico, exibe o gráfico com $plt.show()$. O gráfico resultante ajuda a identificar o "cotovelo", onde a inércia começa a diminuir mais lentamente à medida que mais *clusters* são adicionados.

Execução do Código

```
1 data_path = "/content/drive/My Drive/random_output.csv"
2
3 if not os.path.exists(data_path):
4     raise FileNotFoundError(f"Data path {data_path} does not exist.")
5
6 data = load_and_normalize_data(data_path)
7
8 plot_elbow_method(data, max_k=10)
```

Tabela 5: Execução do Código (K-means e o Método do Cotovelo).

1. Definição do Caminho do Arquivo: Especifica o caminho onde o arquivo CSV está localizado;

2. Verificação da Existência do Arquivo: Usa o*s.path.exists*() para verificar se o caminho do arquivo é válido. E caso não seja, levanta um erro informativo;

3. Carregamento e Normalização dos Dados: Chama a função *load_and_normalize_data* passando o caminho do arquivo e armazena os dados normalizados na variável data;

4. Aplicação do Método do Cotovelo: Chama a função *plot_elbow_method* passando os dados normalizados e um valor máximo de k (10) para encontrar o número ideal de *clusters*.

4.1.3 *Output*

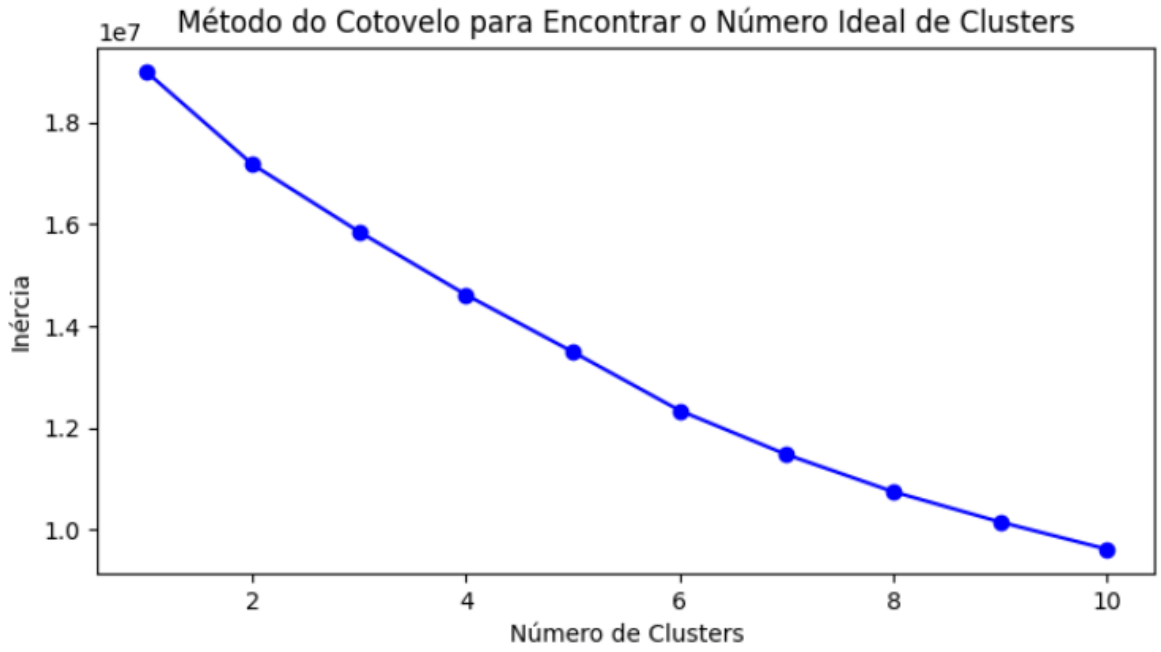

Figura 23: *Output* k-means (método do cotovelo).

Interpretação do gráfico

A Inércia diminui de maneira constante à medida que o número de *clusters* aumenta, sem um ponto de cotovelo muito claro. Isso pode indicar algumas situações:

1. Nenhum Número Ótimo Claro: Não há um ponto de cotovelo claro, o que pode significar que o número ideal de *clusters* não é facilmente determinado apenas pelo método do cotovelo;

2. Dados Complexos: Os dados podem ser complexos ou não estruturados de maneira que não formam *clusters* bem definidos.

4.2 Criação do Código K-NN Classifiers

O K-Nearest Neighbors (K-NN) é um algoritmo de aprendizado supervisionado usado tanto para problemas de classificação quanto de regressão. Ele é particularmente popular devido à sua simplicidade e eficácia em muitos casos práticos.

Como Funciona o K-NN

1. Treinamento: O K-NN é um algoritmo baseado em instâncias, o que significa que não há uma fase de treinamento real. Em vez disso, ele armazena todos os exemplos de treinamento e realiza cálculos na fase de predição;

2. Predição:

    - Classificação: Para classificar um novo ponto de dados, o K-NN calcula a distância entre esse ponto e todos os pontos no conjunto de treinamento, selecionando os K pontos mais próximos (vizinhos). A classe mais frequente entre esses K vizinhos é atribuída ao novo ponto;
    - Regressão: Para prever um valor contínuo, o K-NN calcula a média (ou outro critério) dos valores dos K vizinhos mais próximos.

Passos do Algoritmo K-NN

1. Escolha do Valor de K: Seleciona-se o número de vizinhos K. Um valor pequeno de K pode tornar o modelo sensível ao ruído, enquanto um valor grande pode diluir o impacto de pontos de dados locais;

2. Cálculo da Distância: Calcula-se a distância entre o novo ponto e todos os pontos de treinamento. Distâncias comuns incluem Euclidiana, Manhattan, e Minkowski;

3. Identificação dos Vizinhos: Seleciona-se os K pontos de treinamento mais próximos com base na distância calculada;

4. Classificação ou Regressão:

    - Classificação: Atribui-se a classe mais comum entre os K vizinhos;
    - Regressão: Calcula-se a média dos valores dos K vizinhos.

Vantagens K-NN

- Simplicidade: Fácil de entender e implementar;

- Versatilidade: Pode ser usado para classificação e regressão;
- Sem Treinamento: Não há necessidade de treinamento explícito, tornando-o eficiente em termos de tempo de preparação.

Desvantagens K-NN

- Custo Computacional: Para grandes conjuntos de dados, calcular distâncias para todos os pontos pode ser computacionalmente intensivo;
- Sensibilidade à Escala: A performance pode ser afetada por características com diferentes escalas. Normalização dos dados geralmente é necessária;
- Sensibilidade ao Ruído: Valores atípicos ou ruído nos dados podem afetar significativamente os resultados, especialmente com valores baixos de K.

Aplicações Comuns

- Reconhecimento de Padrões: Como reconhecimento de caracteres e reconhecimento de faces;
- Sistemas de Recomendação: Recomendação de produtos baseados em similaridades de usuários ou itens;
- Detecção de Anomalias: Identificação de padrões anômalos em dados financeiros ou de segurança.

Considerações Finais

A escolha do valor de K é crucial para a performance do K-NN. Normalmente, valores de K ímpares são escolhidos para evitar empates na classificação.

Cross-validation pode ser usada para determinar o melhor valor de K. Além disso, técnicas de redução de dimensionalidade como PCA (*Principal Component Analysis*) podem ser aplicadas para melhorar a eficiência do K-NN em conjuntos de dados de alta dimensionalidade.

4.2.1 Código

```
#K-NN classifiers
import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap
import matplotlib.patches as mpatches
from sklearn import neighbors
from sklearn.preprocessing import StandardScaler

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels

#datasets
test_path = "/content/drive/My Drive/test_dataset.csv"
train_path = "/content/drive/My Drive/train_dataset.csv"

#X and Y
data2, labels2 = load_and_normalize_data(train_path)
data4, labels4 = load_and_normalize_data(test_path)
#classifier object
knn = KNeighborsClassifier(n_neighbors = 5)
#train the classifier
knn.fit(data2, labels2)
#estimate the accuracy
knn.score(data4, labels4)
#predict the response
pred = knn.predict(data4)
#evaluate accuracy
print("Accuracy:", accuracy_score(labels4, pred))
```

Tabela 6: Código K-NN classifiers (primeira parte).

Ao executar o algoritmo com diferentes valores de k, os seguintes resultados de acurácia foram observados:

- $k = 1$: Acurácia de 20.07%

- $k = 5$: Acurácia de 24.16%
- $k = 10$: Acurácia de 25.82%
- $k = 20$: Acurácia de 27.24% (tempo de execução: 3 minutos)
- $k = 50$: Acurácia de 28.55% (tempo de execução: 6 minutos)
- $k = 100$: Acurácia de 29.15% (tempo de execução: 10 minutos)
- $k = 200$: Acurácia de 29.49% (tempo de execução: 18 minutos)

Analisando os resultados, nota-se que, à medida que o valor de k aumenta, há uma ligeira melhora na acurácia do modelo. No entanto, essa melhora é marginal, especialmente para valores mais altos de k. Por exemplo, ao passar de $k = 100$ para $k = 200$, a acurácia aumenta apenas 0.34%, enquanto o custo computacional praticamente dobra, com o tempo de execução aumentando de 10 para 18 minutos.

Esse comportamento destaca uma característica importante do K-NN: embora o aumento de k possa suavizar as fronteiras de decisão e melhorar ligeiramente a acurácia, isso vem a um custo computacional significativo. Para valores muito altos de k, o ganho de acurácia é insuficiente para justificar o aumento no tempo de processamento. Portanto, é crucial encontrar um equilíbrio adequado entre acurácia e eficiência computacional ao escolher o valor de k.

4.2.2 Estrutura do Código (Primeira Parte)

Importação de Bibliotecas

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 from sklearn.model_selection import train_test_split
5 from sklearn.neighbors import KNeighborsClassifier
6 from sklearn.metrics import accuracy_score
7 from matplotlib.colors import ListedColormap
8 import matplotlib.patches as mpatches
9 from sklearn import neighbors
10 from sklearn.preprocessing import StandardScaler
```

Tabela 7: Importação de Bibliotecas (K-NN Classifiers - primeira parte).

- NumPy;
- *Matplotlib.pyplot*;
- Pandas;
- *Prain_test_split*: Função que divide os dados em conjuntos de treinamento e teste (não utilizada neste trecho específico);
- KNeighborsClassifier: Classe que implementa o algoritmo K-NN;
- *Accuracy_score*: Função para calcular a acurácia do modelo;
- ListedColormap e mpatches: Ferramentas para personalização de visualizações (não utilizadas diretamente neste trecho);
- Neighbors: Módulo que contém classes e funções relacionadas a algoritmos de vizinhança (não utilizado diretamente neste trecho);
- StandardScaler.

Função *load_and_normalize_data*

```
def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels
```

Tabela 8: Função *load_and_normalize_data* (K-NN Classifiers - primeira parte).

Descrição da função:

1. Leitura do Arquivo CSV: $pd.read\_csv(file\_path)$ lê um arquivo CSV localizado no caminho especificado e armazena os dados em um DataFrame df;
2. Separação dos Dados e Rótulos:
    - $data = df.iloc[:, :-1].values.astype(np.float32)$: Seleciona todas as colunas, exceto a última, que geralmente contém os rótulos das classes. Os dados são convertidos para o tipo float32 para economizar memória;

- $labels = df['class'].values.astype(np.int64)$: Extrai a coluna *class* como rótulos, convertendo-os para o tipo $int64$.

3. Normalização dos Dados:

   - $scaler = StandardScaler()$: Cria uma instância do StandardScaler, que é usado para normalizar os dados;
   - $data = scaler.fit\_transform(data)$: Ajusta o escalador aos dados e transforma os dados, resultando em uma média de 0 e desvio padrão de 1.

4. Retorno dos Dados Normalizados e Rótulos: A função retorna uma tupla contendo os dados normalizados (*data*) e os rótulos (*labels*).

Carregamento dos Conjuntos de Dados

```
test_path = "/content/drive/My Drive/test_dataset.csv"
train_path = "/content/drive/My Drive/train_dataset.csv"

data2, labels2 = load_and_normalize_data(train_path)
data4, labels4 = load_and_normalize_data(test_path)
```

Tabela 9: Carregamento dos Conjuntos de Dados (K-NN Classifiers - primeira parte).

Descrição:

1. Definição dos Caminhos dos Arquivos: Define variáveis que armazenam os caminhos dos arquivos CSV para os conjuntos de treinamento ($train\_path$) e teste ($test\_path$);

2. Carregamento e Normalização dos Dados:

   - Chama a função $load\_and\_normalize\_data$ duas vezes: uma vez para carregar o conjunto de treinamento ($train\_path$) e outra para o conjunto de teste ($test\_path$).
   - Os dados normalizados são armazenados em *data2* e *data4*, enquanto os rótulos correspondentes são armazenados em *labels2* e *labels4*.

Criação do Classificador K-NN

```
knn = KNeighborsClassifier(n_neighbors=5)
```

Tabela 10: Criação do Classificador K-NN (primeira parte).

Descrição: Cria uma instância do classificador K-NN com 5 vizinhos mais próximos ($n\_neighbors = 5$). Este parâmetro k determina quantos vizinhos serão considerados ao fazer previsões.

Treinamento do Classificador

```
knn.fit(data2, labels2)
```

Tabela 11: Treinamento do Classificador K-NN (primeira parte).

Descrição: Ajusta o modelo K-NN aos dados de treinamento (*data2*) e seus respectivos rótulos (*labels2*). O método fit é responsável por armazenar as informações necessárias sobre os dados para realizar previsões posteriormente.

Avaliação da Acurácia

```
knn.score(data4, labels4)
```

Tabela 12: Avaliação da Acurácia (K-NN Classifiers - primeira parte).

Descrição: O método score avalia o modelo usando o conjunto de teste (*data4*) e seus rótulos correspondentes (*labels4*). Ele retorna a acurácia do modelo, que é a proporção de previsões corretas feitas pelo classificador.

Previsão e Impressão da Acurácia

```
pred = knn.predict(data4)
print("Accuracy:", accuracy_score(labels4, pred))
```

Tabela 13: Previsão e Impressão da Acurácia (K-NN Classifiers - primeira parte).

Descrição:

1. Previsão: $pred = knn.predict(data4)$ usa o modelo treinado para prever os rótulos do conjunto de teste. O resultado é armazenado na variável pred;

2. Cálculo da Acurácia:

- $accuracy\_score(labels4, pred)$: Compara as previsões feitas pelo modelo ($pred$) com os rótulos reais do conjunto de teste ($labels4$) usando a função $accuracy\_score$; O resultado é impresso na tela com a mensagem "Accuracy:", mostrando assim quão bem o classificador se saiu.

4.2.3 Estrutura do Código (Segunda Parte)

Definição do Intervalo de k

```
k_range = range(1, 50)
scores = []
```

Tabela 14: Definição do Intervalo de k (K-NN Classifiers - segunda parte).

Descrição:

1. Definição do Intervalo: $k\_range = range(1, 50)$ cria um objeto range que representa os valores de k que serão testados, variando de 1 a 49. Isso significa que o código irá avaliar o desempenho do classificador K-NN para cada um desses valores de k;

2. Inicialização da Lista de Acurácia: $scores = []$ cria uma lista vazia chamada scores, que será usada para armazenar a acurácia obtida para cada valor de k testado.

Loop para Avaliação da Acurácia

```
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(data2, labels2)
    scores.append(knn.score(data4, labels4))
```

Tabela 15: Loop para Avaliação da Acurácia (K-NN Classifiers - segunda parte).

Descrição:

1. Iteração sobre os Valores de k: O loop for k in k_range itera sobre cada valor de k definido anteriormente;

2. Criação do Classificador K-NN: $knn = KNeighborsClassifier(n\_neighbors = k)$ para cada valor de k, cria uma nova instância do classificador K-NN, especificando o número de vizinhos mais próximos a serem considerados;

3. Treinamento do Classificador: $knn.fit(data2, labels2)$ ajusta o modelo K-NN aos dados de treinamento (*data2*) e seus rótulos (*labels2*). Isso permite que o modelo aprenda a partir dos dados disponíveis;

4. Avaliação da Acurácia: $scores.append(knn.score(data4, labels4))$ avalia a acurácia do modelo usando o conjunto de teste (*data4*) e seus rótulos correspondentes (*labels4*). O resultado é adicionado à lista scores. Assim, ao final do *loop*, *scores* conterá a acurácia para cada valor de k testado.

Visualização dos Resultados

```
plt.figure()
plt.xlabel('k')
plt.ylabel('accuracy')
plt.scatter(k_range, scores)
plt.xticks([0, 10, 20, 30, 40, 50])
```

Tabela 16: Visualização dos Resultados (K-NN Classifiers - segunda parte).

Descrição:

1. Criação da Figura: $plt.figure()$;

2. Rótulos dos Eixos:

   - $plt.xlabel('k')$: Define o rótulo do eixo x como 'k', representando os diferentes valores testados para o parâmetro k;
   - $plt.ylabel('accuracy')$: Define o rótulo do eixo y como '*accuracy*', representando a acurácia correspondente a cada valor de k.

3. Plotagem dos Resultados: $plt.scatter(k\_range, scores)$ cria um gráfico de dispersão onde os valores de k são plotados no eixo x e as respectivas acurácias no eixo y. Isso permite visualizar como a acurácia varia com diferentes escolhas para o parâmetro k;

4. Definição dos Ticks no Eixo x: $plt.xticks([0,\ 10,\ 20,\ 30,\ 40,\ 50])$ define as marcas (*ticks*) no eixo x em intervalos específicos (0, 10, 20, 30, 40 e 50), facilitando a leitura do gráfico.

4.2.4 *Output*

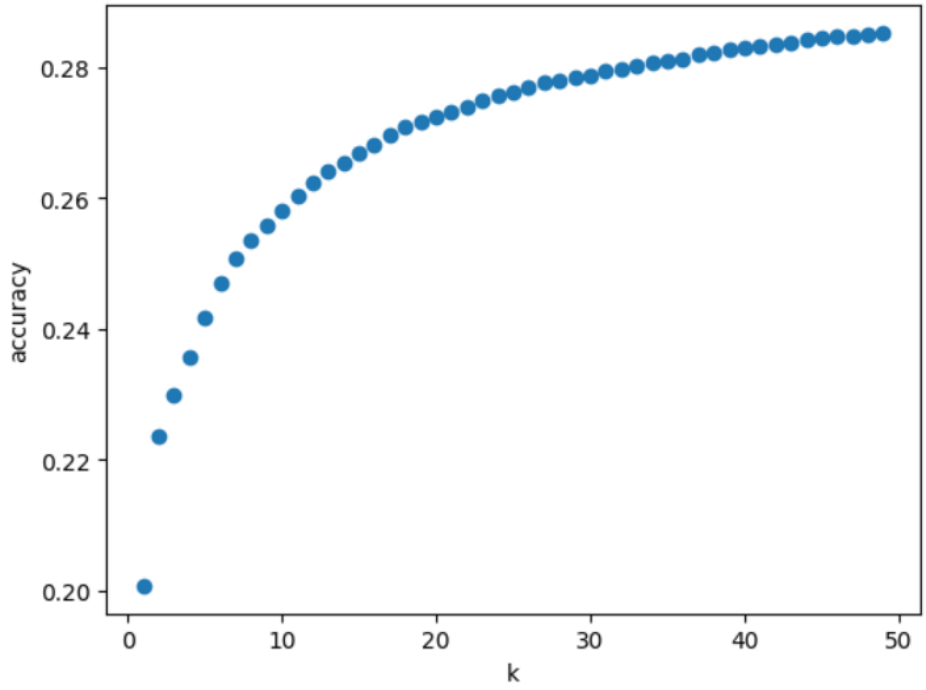

Figura 24: *Output* K-NN.

Interpretação do gráfico

Assim como visto anteriormente, há um aumento do custo computacional e baixo ganho na acurácia. Esse comportamento destaca uma característica importante do K-NN: embora o aumento de k possa suavizar as fronteiras de decisão e melhorar ligeiramente a acurácia, isso vem a um custo computacional significativo.

4.3 Criação do Código Decision Tree

Uma árvore de decisão é um modelo de aprendizado de máquina usado para classificação e regressão. Funciona como um gráfico de decisões onde cada nó interno representa uma "pergunta"sobre uma característica do conjunto de dados, cada ramo representa o resultado dessa pergunta, e cada nó folha representa uma classe ou valor de saída.

1. Construção da Árvore:

    - Escolha da Característica: A construção da árvore começa pela escolha da característica (ou atributo) que melhor divide os dados. Isso é feito usando métricas como ganho de informação (usado no algoritmo ID3), índice Gini (usado no algoritmo CART) ou redução de entropia;

- Divisão do Nó: O conjunto de dados é dividido com base nos valores da característica escolhida. Isso cria ramos na árvore;
- Recursão: Este processo é repetido recursivamente para cada subconjunto resultante, formando novos nós e ramos até que uma das condições de parada seja atingida (por exemplo, todos os dados em um nó têm a mesma classe, ou o número máximo de níveis da árvore é atingido);

2. Predição com a Árvore:

   - Navegação na Árvore: Para fazer uma previsão para uma nova amostra, a amostra é passada pela árvore começando do nó raiz. Em cada nó, uma decisão é tomada com base na característica relevante, seguindo o ramo apropriado até chegar a um nó folha;
   - Classificação ou Regressão: A classe ou valor associado ao nó folha é então usado como a previsão para a amostra.

Importância

1. Interpretação Simples: Árvores de decisão são fáceis de entender e interpretar. Elas espelham processos de tomada de decisão humana, tornando os resultados compreensíveis mesmo para não-especialistas;
2. Manuseio de Dados Categóricos e Numéricos: Árvores de decisão podem lidar com ambos os tipos de dados. Isso as torna versáteis para diferentes tipos de problemas;
3. Pouca Necessidade de Pré-processamento de Dados: Não requerem normalização ou padronização de dados. Além disso, não são afetadas por valores ausentes da mesma forma que outros modelos;
4. Robustez a Outliers: Árvores de decisão são relativamente robustas a outliers porque a divisão é feita com base na maioria dos dados;
5. Capacidade de Capturar Interações Não Lineares: Elas podem capturar relações não lineares entre as características, o que pode ser difícil para modelos lineares.

Desvantagens

1. Sobreajuste (Overfitting): Árvores de decisão tendem a se ajustar demais aos dados de treinamento, especialmente se a árvore for muito profunda;

2. Instabilidade: Pequenas variações nos dados podem resultar em árvores completamente diferentes, o que pode tornar as previsões instáveis;

3. Tendência de Preferência por Atributos com Mais Níveis: Árvores de decisão podem tendem a favorecer características com mais níveis.

Para mitigar algumas dessas desvantagens, métodos como o bagging (por exemplo, Random Forests) ou boosting (por exemplo, Gradient Boosting) são usados, que combinam múltiplas árvores para melhorar a precisão e a robustez das previsões.

4.3.1 Código

```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.colors import ListedColormap, BoundaryNorm
from sklearn import neighbors
import matplotlib.patches as mpatches
import graphviz
from sklearn.tree import export_graphviz
import matplotlib.patches as mpatches
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
import os

def plot_decision_tree(clf, feature_names, class_names, save_path="
    adspy_temp.dot"):
    export_graphviz(clf, out_file=save_path, feature_names=feature_names,
    class_names=class_names, filled=True, impurity=False)
    with open(save_path) as f:
        dot_graph = f.read()
    return graphviz.Source(dot_graph)

def plot_feature_importances(clf, feature_names):
    c_features = len(feature_names)
    plt.barh(range(c_features), clf.feature_importances_)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature name")
```

```
27      plt.yticks(np.arange(c_features), feature_names)
28
29  def load_and_normalize_data(file_path):
30      df = pd.read_csv(file_path)
31      data = df.iloc[:, :-1].values.astype(np.float32)
32      labels = df['class'].values.astype(np.int64)
33      scaler = StandardScaler()
34      data = scaler.fit_transform(data)
35      return data, labels
36
37  # Define the save path for the DOT file
38  save_path = "/content/drive/My Drive/decision_tree.dot"
39
40  # File paths
41  test_path = "/content/drive/My Drive/test_dataset.csv"
42  train_path = "/content/drive/My Drive/train_dataset.csv"
43
44  # Load and normalize data
45  data2, labels2 = load_and_normalize_data(train_path)
46  data4, labels4 = load_and_normalize_data(test_path)
47
48  # Train the classifier
49  clf = DecisionTreeClassifier(max_depth=15, min_samples_leaf=8, random_state
        =0).fit(data2, labels2)
50
51  # Define feature and class names
52  feature_names = ['sensor_1', 'sensor_2', 'sensor_3', 'sensor_4']
53  class_names = [str(i) for i in range(10)]
54
55  # Plot decision tree
56  tree_plot = plot_decision_tree(clf, feature_names, class_names)
57  tree_plot.render("decision_tree")
58
59  # Plot feature importances
60  plot_feature_importances(clf, feature_names)
61  plt.show()
62
63  # Return Accuracy
64  print('Accuracy of DT classifier on training set: {:.2f}'
65        .format(clf.score(data2, labels2)))
```

```
66 print('Accuracy of DT classifier on test set: {:.2f}'
67       .format(clf.score(data4, labels4)))
```

Tabela 17: Código Decision Tree.

4.3.2 Estrutura do Código

Importação de Bibliotecas

```
1  import numpy as np
2  import pandas as pd
3  import seaborn as sn
4  import matplotlib.pyplot as plt
5  import matplotlib.cm as cm
6  from matplotlib.colors import ListedColormap, BoundaryNorm
7  from sklearn import neighbors
8  import matplotlib.patches as mpatches
9  import graphviz
10 from sklearn.tree import export_graphviz
11 from sklearn.tree import DecisionTreeClassifier
12 from sklearn.preprocessing import StandardScaler
13 import os
```

Tabela 18: Importação de Bibliotecas (Decision Tree).

- NumPy;
- Pandas;
- Seaborn: construída sobre o Matplotlib, tem a capacidade de criar gráficos estatísticos;
- Matplotlib.pyplot;
- Graphviz: Biblioteca para criar visualizações gráficas, especialmente útil para visualizar árvores de decisão;
- *export_graphviz*: Função do Scikit-learn que exporta uma árvore de decisão em formato DOT, que pode ser visualizado com Graphviz;
- DecisionTreeClassifier: Classe do Scikit-learn que implementa o algoritmo de árvore de decisão;

- StandardScaler.

Função *plot_decision_tree*

```
def plot_decision_tree(clf, feature_names, class_names, save_path="
    adspy_temp.dot"):
    export_graphviz(clf, out_file=save_path, feature_names=feature_names,
    class_names=class_names, filled=True, impurity=False)
    with open(save_path) as f:
        dot_graph = f.read()
    return graphviz.Source(dot_graph)
```

Tabela 19: Função *plot_decision_tree* (Decision Tree).

Descrição dessa função:

1. Exportação da Árvore - *export_graphviz*(...): Exporta a árvore de decisão treinada (clf) para um arquivo DOT. Os parâmetros incluem:
   - out_file: caminho onde o arquivo DOT será salvo;
   - $feature\_names$: nomes das características usadas na árvore;
   - $class\_names$: nomes das classes alvo;
   - $filled = True$: preenche os nós com cores baseadas na classe predominante;
   - $impurity = False$: não exibe a impureza dos nós.
2. Lê o arquivo DOT gerado e armazena seu conteúdo na variável *dot_graph*;
3. Retorna uma instância do objeto *graphviz.Source*, que pode ser usado para renderizar a árvore visualmente.

Função *plot_feature_importances*

```
def plot_feature_importances(clf, feature_names):
    c_features = len(feature_names)
    plt.barh(range(c_features), clf.feature_importances_)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature name")
    plt.yticks(np.arange(c_features), feature_names)
```

Tabela 20: Função *plot_feature_importances* (Decision Tree).

Descrição dessa função:

1. $c\_features = len(feature\_names)$: Conta o número total de características;
2. Criação do Gráfico de Importância das Características
   - $plt.barh(...)$: Plota um gráfico horizontal onde cada barra representa a importância da característica correspondente na árvore de decisão ($clf.feature\_importances\_$);
   - Define rótulos para os eixos x e y e ajusta os ticks no eixo y para mostrar os nomes das características.

Função *load_and_normalize_data*

```
def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels
```

Tabela 21: Função *load_and_normalize_data* (Decision Tree).

Descrição dessa função:

1. Lê um arquivo CSV e armazena os dados em um DataFrame;
2. Extrai todas as colunas exceto a última como dados e a coluna 'class' como rótulos;
3. Usa o *StandardScaler* para normalizar os dados;
4. Retorna uma tupla contendo os dados normalizados e os rótulos.

Carregamento dos Conjuntos de Dados

```
# Define the save path for the DOT file
save_path = "/content/drive/My Drive/decision_tree.dot"

# File paths
test_path = "/content/drive/My Drive/test_dataset.csv"
train_path = "/content/drive/My Drive/train_dataset.csv"
```

```
7
8 # Load and normalize data
9 data2, labels2 = load_and_normalize_data(train_path)
10 data4, labels4 = load_and_normalize_data(test_path)
```

Tabela 22: Carregamento dos Conjuntos de Dados (Decision Tree).

Descrição:

1. Define variáveis que armazenam os caminhos dos arquivos CSV para conjuntos de treinamento e teste;
2. Chama a função *load_and_normalize_data* duas vezes para carregar e normalizar ambos os conjuntos.

Treinamento do Classificador

```
1 clf = DecisionTreeClassifier(max_depth=15, min_samples_leaf=8, random_state
    =0).fit(data2, labels2)
```

Tabela 23: Treinamento do Classificador (Decision Tree).

Descrição:

- $max\_depth = 15$: Limita a profundidade máxima da árvore a 15 níveis;
- $min\_samples\_leaf = 8$: Define que um nó deve ter pelo menos 8 amostras para ser considerado como folha;
- $random\_state = 0$: Garante reprodutibilidade nos resultados;
- O método $.fit(...)$ treina o classificador com os dados normalizados e seus rótulos.

Definição dos Nomes das Características e Classes

```
1 # Define feature and class names
2 feature_names = ['sensor_1', 'sensor_2', 'sensor_3', 'sensor_4']
3 class_names = [str(i) for i in range(10)]
```

Tabela 24: Definição dos Nomes das Características e Classes (Decision Tree).

Visualização da Árvore de Decisão

```
# Plot decision tree
tree_plot = plot_decision_tree(clf, feature_names, class_names)
tree_plot.render("decision_tree")
```

Tabela 25: Visualização da Árvore de Decisão (Decision Tree).

Descrição:

1. Chama a função *plot_decision_tree*, passando o classificador treinado (clf), os nomes das características (*feature_names*) e os nomes das classes (*class_names*);

2. O método *.render("decision_tree")* gera o arquivo visual da árvore no formato especificado (DOT).

Visualização das Importâncias das Características

```
# Plot feature importances
plot_feature_importances(clf, feature_names)
plt.show()
```

Tabela 26: Visualização das Importâncias das Características (Decision Tree).

Avaliação da Acurácia

```
# Return Accuracy
print('Accuracy of DT classifier on training set: {:.2f}'.format(clf.score(
    data2, labels2)))
print('Accuracy of DT classifier on test set: {:.2f}'.format(clf.score(
    data4, labels4)))
```

Tabela 27: Avaliação da Acurácia (Decision Tree).

1. Usa o método .score(...) no conjunto de treinamento (data2, labels2) para calcular a acurácia do modelo;

2. Faz o mesmo cálculo para o conjunto de teste (data4, labels4);

3. Imprime a acurácia obtida em ambos os conjuntos formatada com duas casas decimais.

4.3.3 *Output*

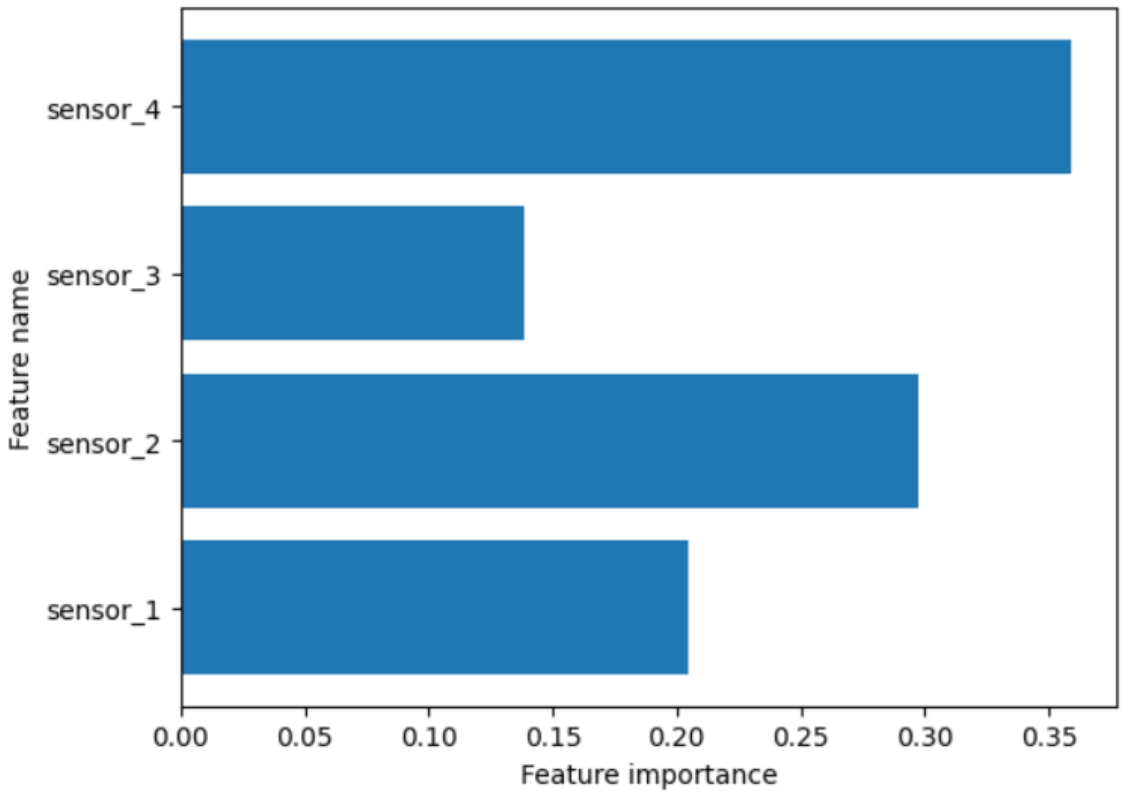

Figura 25: *Output* Decision Tree.

A acurácia do classificador de Árvore de Decisão (DT) no conjunto de treinamento foi de 0.30, enquanto no conjunto de teste foi de 0.29. Esses resultados indicam que, mesmo com um modelo mais simples, a capacidade de generalização da árvore é limitada e a diferença mínima entre as acurácias do treinamento e teste sugere que o modelo não está sofrendo de *overfitting*, mas sim de um baixo poder de predição.

Além disso, foi observado que ao aumentar a profundidade máxima da árvore de decisão para valores acima de $max\_depth = 15$, o custo computacional começa a crescer rapidamente. Para profundidades muito altas, o tempo de processamento pode superar 5 horas, conforme o modelo se torna mais complexo e requer maior capacidade de cálculo.

4.4 Criação do Código Scatter Maps

O *scatter plot matrix* (ou matriz de gráficos de dispersão) é uma ferramenta visual utilizada principalmente para explorar a relação entre múltiplas variáveis em um conjunto de dados e também é utilizado para visualizar padrões, tendências e *outliners*. Além dos gráficos de dispersão, a matriz pode incluir histogramas ou gráficos de densidade ao longo

da diagonal principal para mostrar a distribuição de cada variável individualmente.

```
#C digo Scatter Maps

import os
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from pandas.plotting import scatter_matrix

def load_and_normalize_data(data_path):
    data = pd.read_csv(data_path)
    labels = data.pop('class')
    normalized_data = (data - data.mean()) / data.std()
    return normalized_data, labels

data_path = "/content/drive/My Drive/random_output.csv"

if not os.path.exists(data_path):
    raise FileNotFoundError(f"Data path {data_path} does not exist.")

# Define a colormap with more distinguishable colors
colors = ['#FF0000', '#0000FF', '#00FF00', '#FF00FF', '#00FFFF', '#FFFF00',
    '#FFA500', '#800080', '#008000', '#FFC0CB']
cmap = ListedColormap(colors)

data1, labels1 = load_and_normalize_data(data_path)

# Create scatter matrix
scatter = scatter_matrix(data1, c=labels1, marker='o', s=40, hist_kwds={'
    bins': 15}, figsize=(10, 10), cmap=cmap)

# Add legend
handles = [plt.Line2D([], [], color=cmap(i), marker='o', linestyle='',
    markersize=10) for i in range(len(labels1.unique()))]
labels = [f'Class {i}' for i in labels1.unique()]
plt.legend(handles, labels, loc='upper right', bbox_to_anchor=(1.2, 1),
    title='Classes')

plt.show()
```

Tabela 28: Código Scatter Maps.

4.4.1 Estrutura do Código

Importação de Bibliotecas

```
#Scatter Maps 3D

import os
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D

def load_and_normalize_data(data_path):
    data = pd.read_csv(data_path)
    labels = data.pop('class')
    normalized_data = (data - data.mean()) / data.std()
    return normalized_data, labels

data_path = "/content/drive/My Drive/random_output.csv"

if not os.path.exists(data_path):
    raise FileNotFoundError(f"Data path {data_path} does not exist.")

# Define a colormap with more distinguishable colors
colors = ['#FF0000', '#0000FF', '#00FF00', '#FF00FF', '#00FFFF', '#FFFF00',
    '#FFA500', '#800080', '#008000', '#FFC0CB']
cmap = ListedColormap(colors)

data, labels = load_and_normalize_data(data_path)

# Selecionando tr s vari veis para o gr fico 3D
x_var = 'sensor_2'
y_var = 'sensor_3'
z_var = 'sensor_4'

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Criando o scatter plot 3D
scatter = ax.scatter(data[x_var], data[y_var], data[z_var], c=labels, cmap=
```

```
    cmap, marker='o', s=40)

# Adicionando r tulos
ax.set_xlabel(x_var)
ax.set_ylabel(y_var)
ax.set_zlabel(z_var)

# Adicionando legenda
handles = [plt.Line2D([], [], color=cmap(i), marker='o', linestyle='',
    markersize=10) for i in range(len(labels.unique()))]
label_classes = [f'Class {i}' for i in labels.unique()]
ax.legend(handles, label_classes, loc='upper left', bbox_to_anchor=(1.2, 1)
    , title='Classes')

plt.show()
```

Tabela 29: Importação de Bibliotecas (Scatter Maps).

- OS;
- Pandas;
- Matplotlib.pyplot;
- ListedColormap: Classe do matplotlib que permite criar um mapa de cores personalizado;
- *Scatter_matrix*: Função do pandas que cria uma matriz de gráficos de dispersão.

Função *load_and_normalize_data*

```
def load_and_normalize_data(data_path):
    data = pd.read_csv(data_path)
    labels = data.pop('class')
    normalized_data = (data - data.mean()) / data.std()
    return normalized_data, labels
```

Tabela 30: Função *load_and_normalize_data* (Scatter Maps).

Descrição da função:

1. Leitura do arquivo CSV: $data = pd.read\_csv(data\_path)$ lê um arquivo CSV localizado no caminho especificado ($data\_path$) e armazena os dados em um *DataFrame* "*data*";

2. Separação dos Rótulos: $labels = data.pop('class')$ remove a coluna '*class*' do *DataFrame* e armazena seus valores na variável *labels*. Essa coluna é assumida como a variável alvo ou classe;

3. Normalização dos Dados: $normalized\_data = (data - data.mean())/data.std()$ normaliza os dados subtraindo a média e dividindo pelo desvio padrão, resultando em dados com média 0 e desvio padrão 1;

4. Retorno dos Dados Normalizados e Rótulos: A função retorna uma tupla contendo os dados normalizados ($normalized\_data$) e os *labels*.

Verificação da Existência do Arquivo

```
data_path = "/content/drive/My Drive/random_output.csv"

if not os.path.exists(data_path):
    raise FileNotFoundError(f"Data path {data_path} does not exist.")
```

Tabela 31: Verificação da Existência do Arquivo (Scatter Maps).

Descrição:

1. Define a variável $data\_path$ que contém o caminho para o arquivo CSV;

2. Usa $os.path.exists(data\_path)$ para verificar se o arquivo existe. Se não existir, levanta um erro informativo *FileNotFoundError*.

Definição do Mapa de Cores

```
colors = ['#FF0000', '#0000FF', '#00FF00', '#FF00FF', '#00FFFF', '#FFFF00',
    '#FFA500', '#800080', '#008000', '#FFC0CB']
cmap = ListedColormap(colors)
```

Tabela 32: Definição do Mapa de Cores (Scatter Maps).

Descrição:

1. Cria uma lista chamada *colors* com códigos hexadecimais representando cores distintas que serão usadas para diferenciar as classes no gráfico;

2. $cmap = ListedColormap(colors)$ cria um objeto *ListedColormap* usando as cores definidas, que será utilizado para colorir os pontos no gráfico.

Carregamento e Normalização dos Dados

```
data1, labels1 = load_and_normalize_data(data_path)
```

Tabela 33: Carregamento e Normalização dos Dados (Scatter Maps).

Descrição: Chama a função *load_and_normalize_data*, passando o caminho do arquivo CSV (*data_path*). Os dados normalizados são armazenados em *data1*, enquanto os rótulos correspondentes são armazenados em *labels1*.

Criação da Matriz de Gráficos de Dispersão

```
scatter = scatter_matrix(data1, c=labels1, marker='o', s=40, hist_kwds={'
    bins': 15}, figsize=(10, 10), cmap=cmap)
```

Tabela 34: Criação da Matriz de Gráficos de Dispersão (Scatter Maps).

Descrição:

1. A função *scatter_matrix* gera uma matriz de gráficos de dispersão para todas as combinações possíveis das colunas em *data1*;

2. Os parâmetros utilizados incluem:

   - $c = labels1$: Define as cores dos pontos com base nos rótulos das classes;
   - $marker = 'o'$: Especifica o formato dos marcadores como círculos;
   - $s = 40$: Define o tamanho dos marcadores;
   - $hist\_kwds = 'bins' : 15$: Especifica que os histogramas nas diagonais devem ter 15 *bins*;
   - $figsize = (10, 10)$: Define o tamanho da figura como 10x10 polegadas;
   - $cmap = cmap$: Aplica o mapa de cores definido anteriormente.

Adição da Legenda

```
handles = [plt.Line2D([], [], color=cmap(i), marker='o', linestyle='',
    markersize=10) for i in range(len(labels1.unique()))]
labels = [f'Class {i}' for i in labels1.unique()]
plt.legend(handles, labels, loc='upper right', bbox_to_anchor=(1.2, 1),
    title='Classes')
```

Tabela 35: Adição da Legenda (Scatter Maps).

Descrição:

1. A lista *handles* é criada usando uma compreensão de lista que gera objetos *Line2D* para cada classe única nos rótulos (*labels1*). Cada objeto representa um ponto na legenda com a cor correspondente;

2. A lista *labels* é criada contendo strings que representam cada classe (por exemplo, '*Class 0*', '*Class 1*', etc.);

3. O método *plt.legend*(...) adiciona a legenda ao gráfico na posição especificada ($loc = '$ *upperright*$'$) e ajusta sua posição com o parâmetro *bbox_to_anchor*.

Exibição do Gráfico

```
plt.show()
```

Tabela 36: Exibição do Gráfico (Scatter Maps).

Descrição: O método *plt.show*() exibe todos os gráficos criados até este ponto na tela.

4.4.2 *Output*

Distribuições Individuais

Os histogramas na diagonal principal mostram a distribuição de cada sensor individualmente. Pode-se ver que *sensor*_1, *sensor*_2 e *sensor*_3 têm distribuições concentradas ao redor de um valor central, enquanto *sensor*_4 tem uma distribuição mais espalhada.

Classes

As cores diferentes representam diferentes classes, conforme mostrado na legenda à direita. Há uma clara diferenciação de algumas classes em relação a *sensor*_4. As classes 7 (verde claro) e 3 (azul escuro) são bem distintas das outras classes neste sensor.

Overlapping e Separação de Classes

Em muitos dos gráficos de dispersão, há uma sobreposição significativa entre as classes, indicando que pode ser difícil separar essas classes usando apenas essas duas variáveis. No entanto, em gráficos que envolvem $sensor\_4$, especialmente contra $sensor\_1$, $sensor\_2$, e $sensor\_3$, algumas classes mostram menos sobreposição, sugerindo que $sensor\_4$ pode ser um bom discriminador para algumas classes.

Inferências específicas

Sensor 4

Tal sensor parece ser particularmente útil para distinguir classes. Por exemplo, as 7 (verde claro) e 3 (azul escuro) são muito distintas em relação a $sensor\_4$. Porém, ainda é possível observar sobreposições de classe, como a classe amarela e magenta tendo distribuições próximas à classe azul escuro. $sensor\_4$ contra $sensor\_1$, $sensor\_2$ e $sensor\_3$ mostra menos sobreposição para algumas classes, o que pode indicar uma capacidade discriminativa mais alta.

Correlações e Distribuições

A maioria das relações entre os outros sensores ($sensor\_1$, $sensor\_2$ e $sensor\_3$) não mostra padrões claros de correlação positiva ou negativa. A dispersão parece ser mais uniforme, sugerindo baixa correlação entre essas variáveis.

Outliers

Existem alguns pontos que estão distantes do agrupamento central, indicando possíveis outliers. Esses outliers podem ser importantes para investigações adicionais ou pré-processamento de dados.

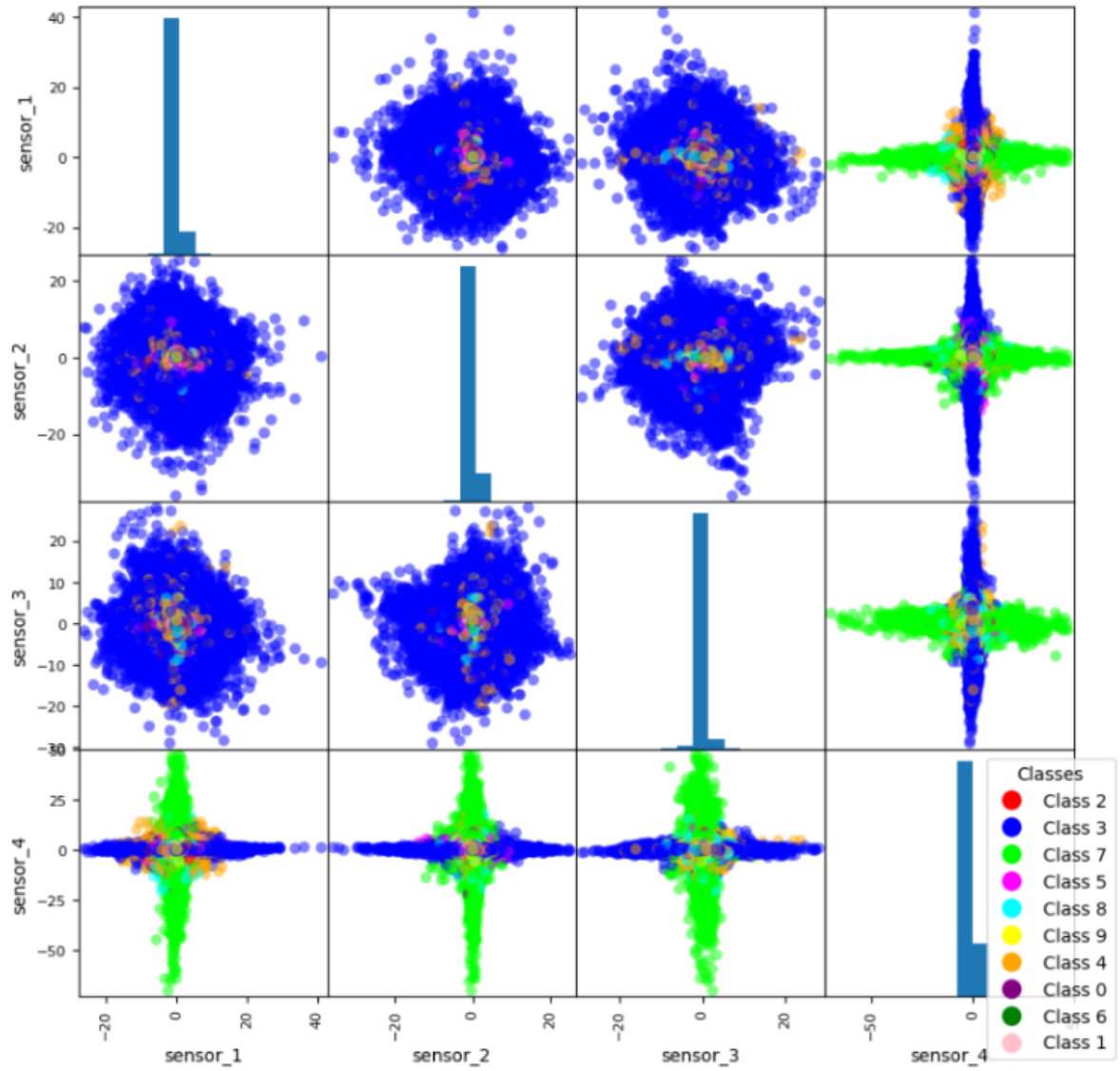

Figura 26: *Output* Scatter Maps.

4.5 Criação do Código Scatter Maps 3D

A motivação desse código é observar melhor a relação dos 3 sensores na classificação de cada movimento.

```
#Scatter Maps 3D

import os
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D

def load_and_normalize_data(data_path):
```

```
    data = pd.read_csv(data_path)
    labels = data.pop('class')
    normalized_data = (data - data.mean()) / data.std()
    return normalized_data, labels

data_path = "/content/drive/My Drive/random_output.csv"

if not os.path.exists(data_path):
    raise FileNotFoundError(f"Data path {data_path} does not exist.")

# Define a colormap with more distinguishable colors
colors = ['#FF0000', '#0000FF', '#00FF00', '#FF00FF', '#00FFFF', '#FFFF00',
    '#FFA500', '#800080', '#008000', '#FFC0CB']
cmap = ListedColormap(colors)

data, labels = load_and_normalize_data(data_path)

# Selecionando tr s vari veis para o gr fico 3D
x_var = 'sensor_2'
y_var = 'sensor_3'
z_var = 'sensor_4'

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Criando o scatter plot 3D
scatter = ax.scatter(data[x_var], data[y_var], data[z_var], c=labels, cmap=
    cmap, marker='o', s=40)

# Adicionando r tulos
ax.set_xlabel(x_var)
ax.set_ylabel(y_var)
ax.set_zlabel(z_var)

# Adicionando legenda
handles = [plt.Line2D([], [], color=cmap(i), marker='o', linestyle='',
    markersize=10) for i in range(len(labels.unique()))]
label_classes = [f'Class {i}' for i in labels.unique()]
ax.legend(handles, label_classes, loc='upper left', bbox_to_anchor=(1.2, 1)
    , title='Classes')
```

```
46
47 plt.show()
```

Tabela 37: Código Scatter Maps 3D.

4.5.1 *Output*

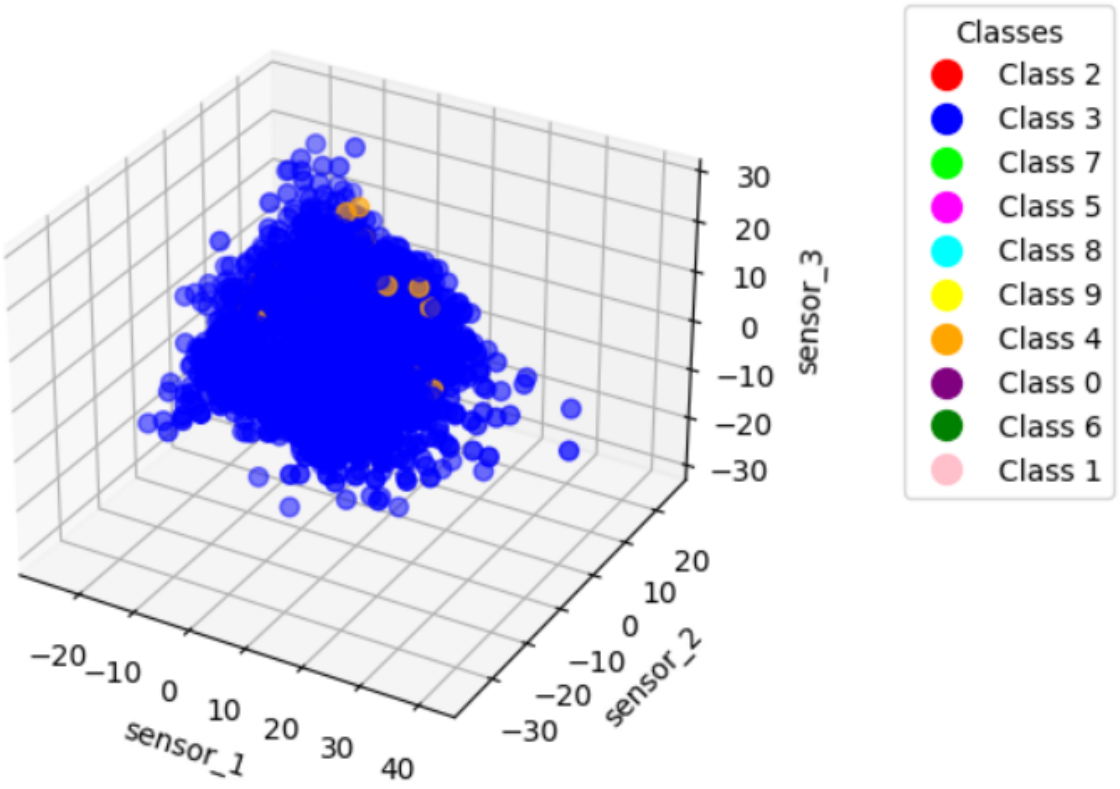

Figura 27: *Output* Scatter Maps 3D sensores 1, 2 e 3.

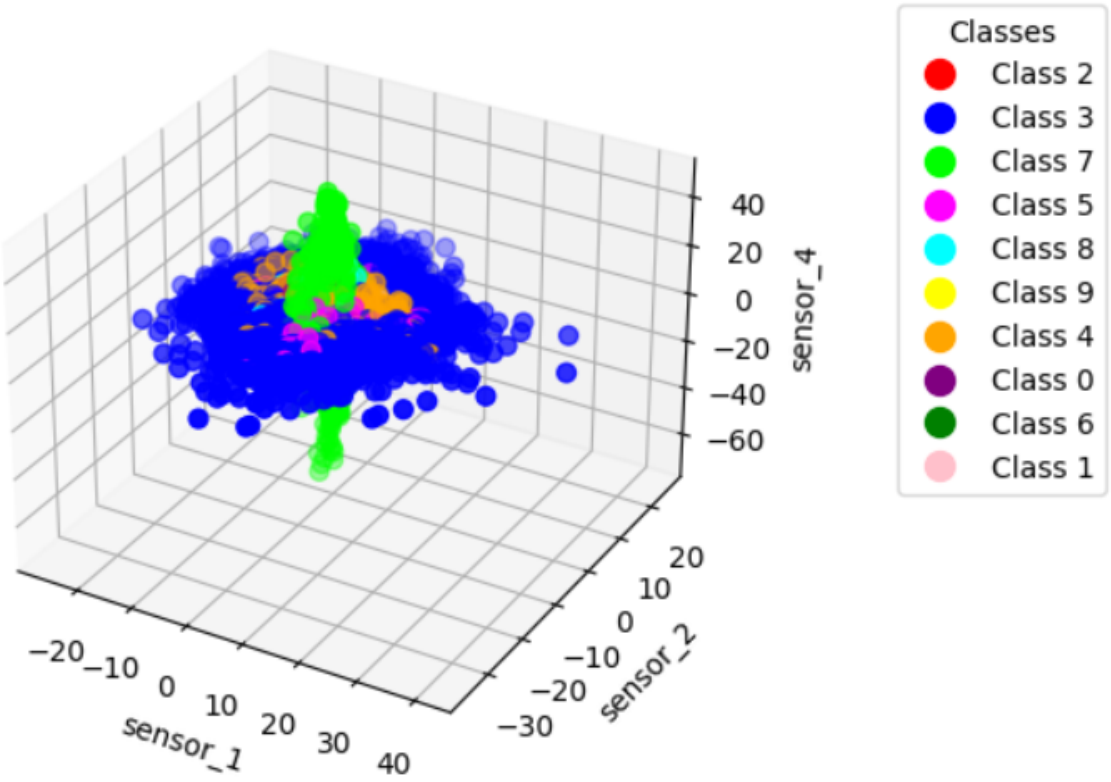

Figura 28: *Output* Scatter Maps 3D sensores 1, 2 e 4.

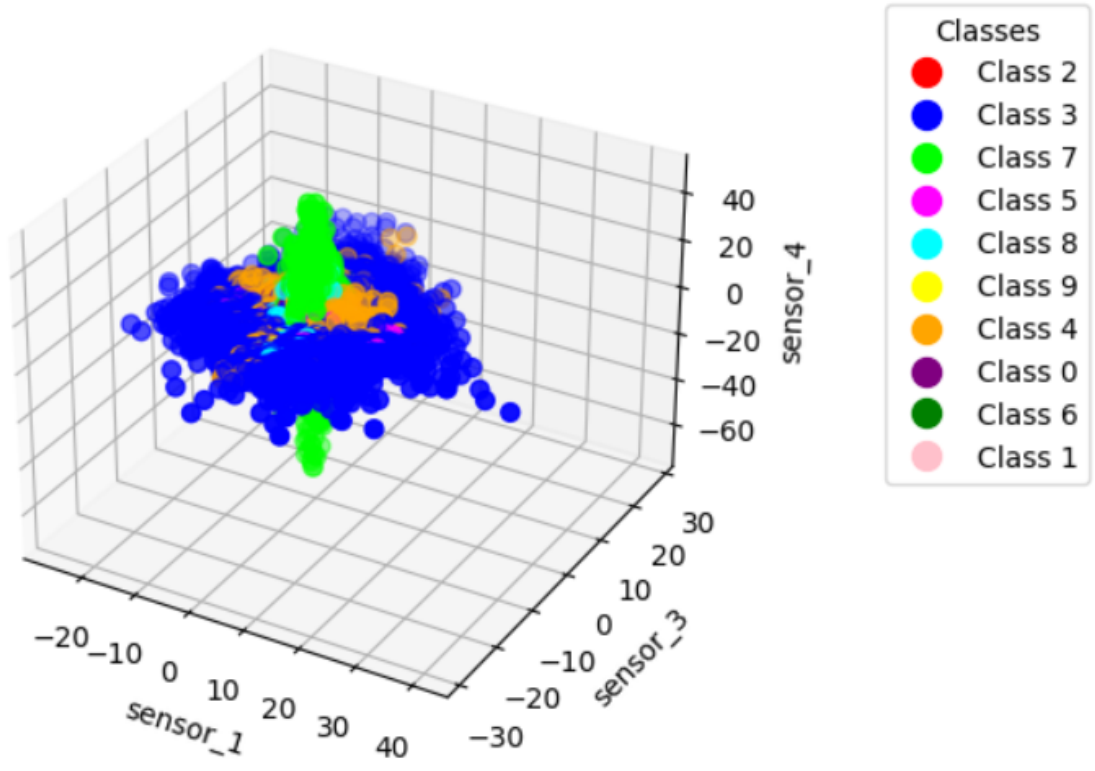

Figura 29: *Output* Scatter Maps 3D sensores 1, 3 e 4.

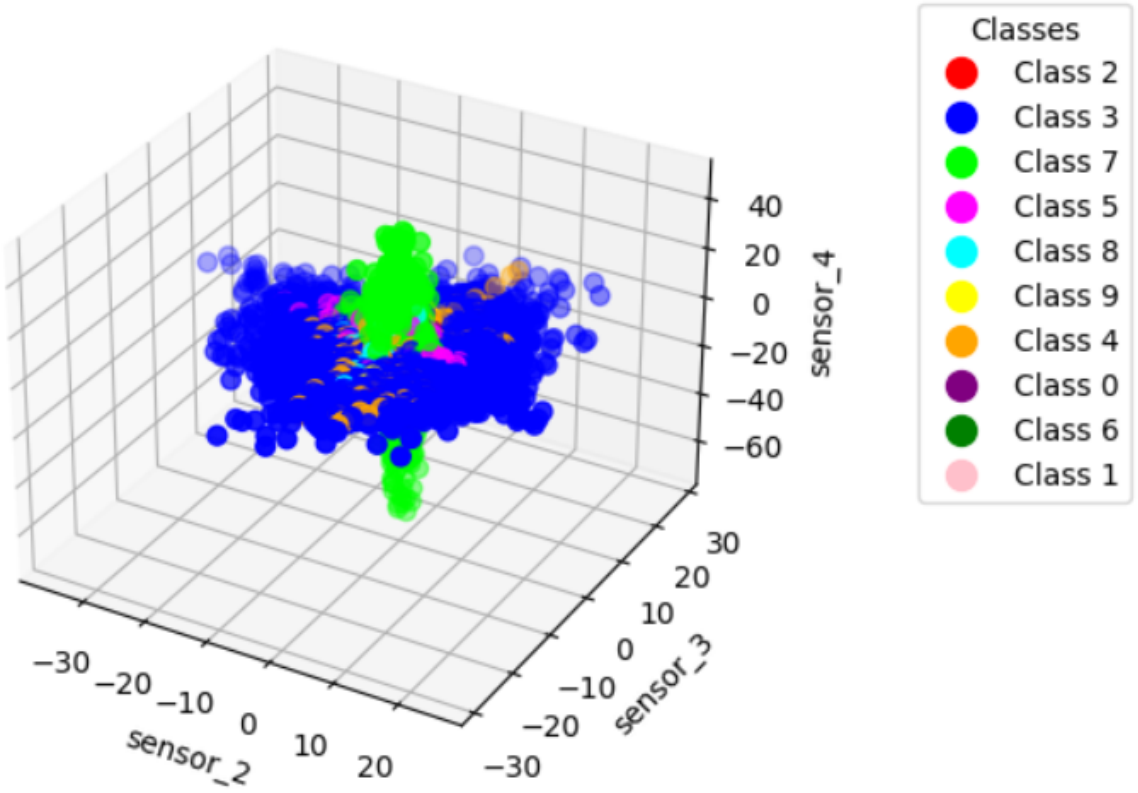

Figura 30: *Output* Scatter Maps 3D sensores 2, 3 e 4.

Reforça que o sensor 4 é o que mais influencia no gráfico, com o sensor 3 é possível se ver mais cores e o sensor 1 é ligeiramente mais expressivo que o 2. Isto é, o sensor 2 é o que menos influencia nos dados.

4.6 Criação do código Matriz de Confusão Multi-Classe com K-NN

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
```

```
14      scaler = StandardScaler()
15      data = scaler.fit_transform(data)
16      return data, labels
17
18  def train_and_evaluate_knn(X_train, y_train, X_test, y_test):
19      # Treinamento com KNN
20      knn = KNeighborsClassifier(n_neighbors=200).fit(X_train, y_train)
21      knn_predicted = knn.predict(X_test)
22
23      # Avalia  o
24      confusion_mc = confusion_matrix(y_test, knn_predicted)
25      accuracy = accuracy_score(y_test, knn_predicted)
26
27      df_cm = pd.DataFrame(confusion_mc, index=range(10), columns=range(10))
28
29      # Aumentando a  rea  do gr fico para 10x8
30      plt.figure(figsize=(10, 8))
31      sns.heatmap(df_cm, annot=True, fmt='d', cmap='Blues', cbar=False)
32      plt.title(f'KNN Classifier\nAccuracy: {accuracy:.3f}')
33      plt.ylabel('True label')
34      plt.xlabel('Predicted label')
35      plt.show()
36
37  # Caminhos dos datasets
38  test_path = "/content/drive/My Drive/test_dataset.csv"
39  train_path = "/content/drive/My Drive/train_dataset.csv"
40
41  # Carregar e normalizar os dados
42  X_train, y_train = load_and_normalize_data(train_path)
43  X_test, y_test = load_and_normalize_data(test_path)
44
45  # Treinar e avaliar o modelo KNN
46  train_and_evaluate_knn(X_train, y_train, X_test, y_test)
```

Tabela 38: Código Matriz de Confusão Multi-Classe com K-NN.

4.6.1 *Output* Inicial

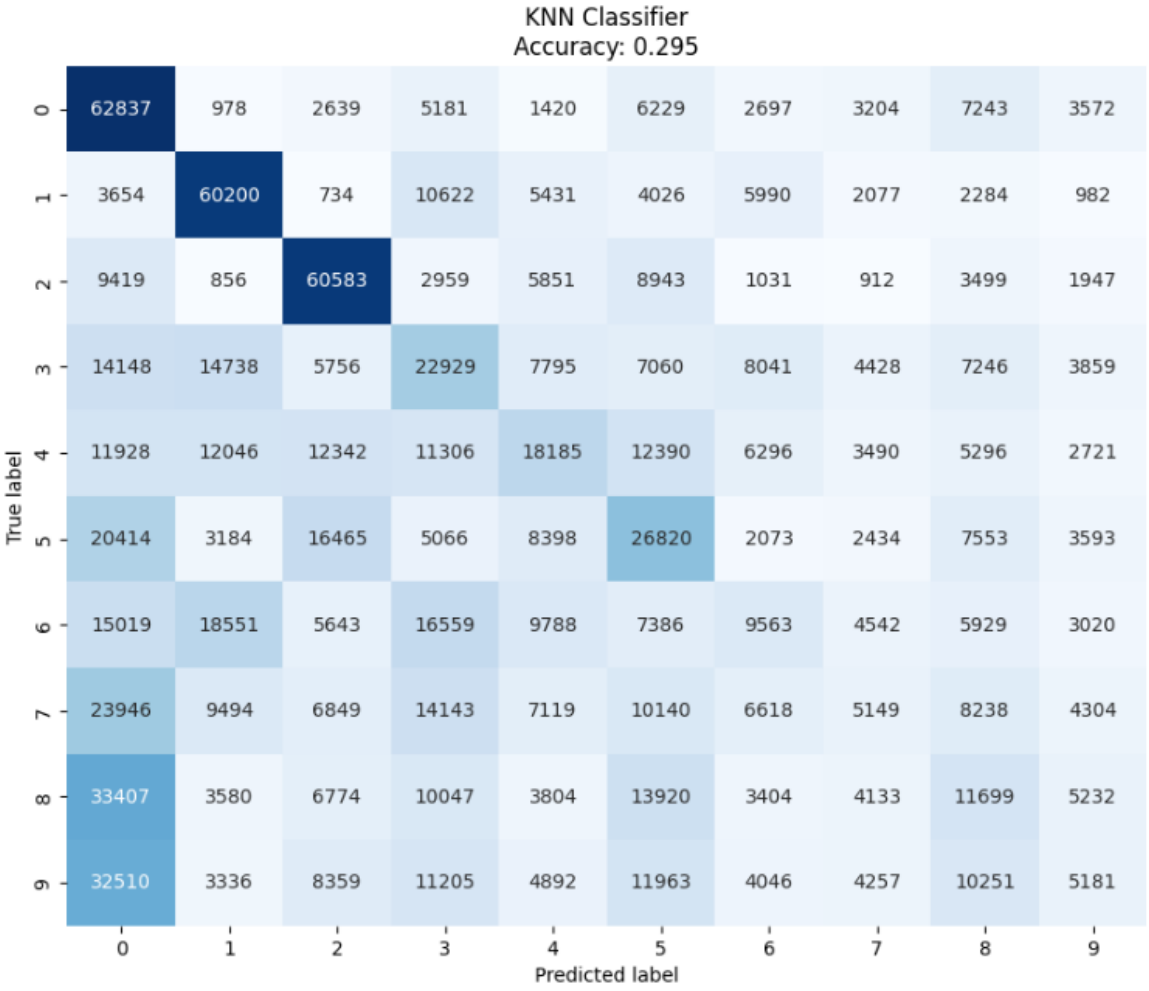

Figura 31: *Output* Inicial da Matriz de Confusão Multi-Classe com K-NN.

4.6.2 Aperfeiçoando a Visão dos Dados

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
```

```
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels

def train_and_evaluate_knn(X_train, y_train, X_test, y_test):
    # Treinamento com KNN
    knn = KNeighborsClassifier(n_neighbors=200).fit(X_train, y_train)
    knn_predicted = knn.predict(X_test)

    # Avalia  o
    confusion_mc = confusion_matrix(y_test, knn_predicted)
    accuracy = accuracy_score(y_test, knn_predicted)
    precision = precision_score(y_test, knn_predicted, average='macro')
    recall = recall_score(y_test, knn_predicted, average='macro')
    f1 = f1_score(y_test, knn_predicted, average='macro')

    print(f"Acur cia total: {accuracy:.3f}")
    print(f"Precis o total: {precision:.3f}")
    print(f"Recall total: {recall:.3f}")
    print(f"F1 Score total: {f1:.3f}")

    # Calcular as m tricas por classe
    class_precision = precision_score(y_test, knn_predicted, average=None)
    class_recall = recall_score(y_test, knn_predicted, average=None)
    class_f1 = f1_score(y_test, knn_predicted, average=None)

    for i in range(len(class_precision)):
        print(f"\nClasse {i}:")
        print(f"  Acur cia: {confusion_mc[i, i] / confusion_mc.sum(axis=1)
    [i]:.3f}")
        print(f"  Precis o: {class_precision[i]:.3f}")
        print(f"  Recall: {class_recall[i]:.3f}")
        print(f"  F1 Score: {class_f1[i]:.3f}")

    df_cm = pd.DataFrame(confusion_mc, index=range(10), columns=range(10))

    # Aumentando a  rea  do gr fico para 10x8
    plt.figure(figsize=(10, 8))
    sns.heatmap(df_cm, annot=True, fmt='d', cmap='Blues', cbar=False)
```

```
    plt.title(f'KNN Classifier\nAccuracy: {accuracy:.3f}')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

# Caminhos dos datasets
test_path = "/content/drive/My Drive/test_dataset.csv"
train_path = "/content/drive/My Drive/train_dataset.csv"

# Carregar e normalizar os dados
X_train, y_train = load_and_normalize_data(train_path)
X_test, y_test = load_and_normalize_data(test_path)

# Treinar e avaliar o modelo KNN
train_and_evaluate_knn(X_train, y_train, X_test, y_test)
```

Tabela 39: Código Matriz de Confusão Multi-Classe com K-NN.

4.6.3 *Output*

A alteração no código possibilitou a análise dos parâmetros de cada movimento, como listado abaixo.

1. **Modelo:** acurácia de 29,5%, precisão de 26%, *recall* de 29,5% e *F1 score* de 26,1%.
2. **Posição neutra:** acurácia de 65,5%, precisão de 27,6%, *recall* de 65,5% e *F1 score* de 38,9%.
3. **Extensão de pulso:** acurácia de 62,7%, precisão de 47,4%, *recall* de 62,7% e *F1 score* de 54%.
4. **Flexão de pulso:** acurácia de 63,1%, precisão de 48%, *recall* de 63,1% e *F1 score* de 54,5%.
5. **Desvio ulnar:** acurácia de 23,9%, precisão de 20,8%, *recall* de 23,9% e *F1 score* de 22,3%.
6. **Desvio radial:** acurácia de 18,9%, precisão de 25%, *recall* de 18,9% e *F1 score* de 21,6%.
7. **Punho fechado:** acurácia de 27,9%, precisão de 24,6%, *recall* de 27,9% e *F1 score* de 26,2%.

8. **Abdução dos dedos:** acurácia de 10%, precisão de 19,2%, *recall* de 10% e *F1 score* de 13,1%.

9. **Adução dos dedos:** acurácia de 5,4%, precisão de 14,9%, *recall* de 5,4% e *F1 score* de 7,9%.

10. **Supinação:** acurácia de 12,2%, precisão de 16,9%, *recall* de 12,2% e *F1 score* de 14,2%.

11. **Pronação:** acurácia de 5,4%, precisão de 15,1%, *recall* de 5,4% e *F1 score* de 7,9%.

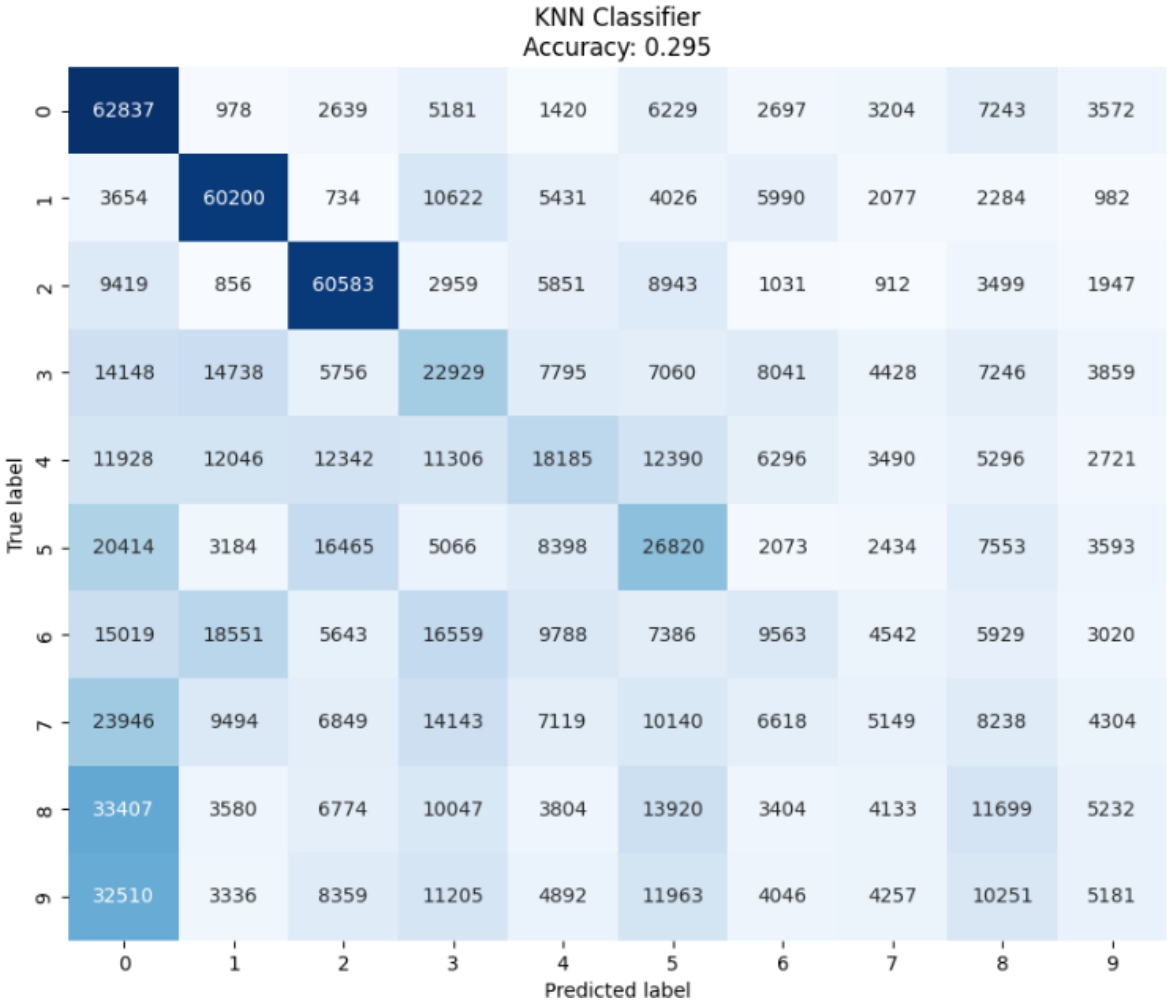

Figura 32: *Output* Matriz de Confusão Multi-Classe com K-NN.

Com a melhoria da visualização da saída foi possível notar que as 4 últimas classes se destacam pela baixa acurácia, enquanto as 3 primeiras possuem alta acurácia, porém, baixa precisão, indicando que outros movimentos são confundidos com esses. Agora, excluindo essas 3 da comparação, as demais indicam baixo aprendizado de suas características.

4.7 Criação do código Matriz de Confusão Multi-Classe com DT

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels

def train_and_evaluate_decision_tree(X_train, y_train, X_test, y_test):
    # Training with Decision Tree
    clf = DecisionTreeClassifier(max_depth=15, min_samples_leaf=8,
    random_state=0).fit(X_train, y_train)
    dt_predicted = clf.predict(X_test)

    # Evaluation
    confusion_mc = confusion_matrix(y_test, dt_predicted)
    accuracy = accuracy_score(y_test, dt_predicted)
    precision = precision_score(y_test, dt_predicted, average='macro')
    recall = recall_score(y_test, dt_predicted, average='macro')
    f1 = f1_score(y_test, dt_predicted, average='macro')

    print(f"Acur cia total: {accuracy:.3f}")
    print(f"Precis o total: {precision:.3f}")
    print(f"Recall total: {recall:.3f}")
    print(f"F1 Score total: {f1:.3f}")

    # Calculate metrics per class
    class_precision = precision_score(y_test, dt_predicted, average=None)
    class_recall = recall_score(y_test, dt_predicted, average=None)
```

```
    class_f1 = f1_score(y_test, dt_predicted, average=None)

    for i in range(len(class_precision)):
        print(f"\nClasse {i}:")
        print(f"  Acur cia: {confusion_mc[i, i] / confusion_mc.sum(axis=1)
    [i]:.3f}")
        print(f"  Precis o: {class_precision[i]:.3f}")
        print(f"  Recall: {class_recall[i]:.3f}")
        print(f"  F1 Score: {class_f1[i]:.3f}")

    df_cm = pd.DataFrame(confusion_mc, index=range(10), columns=range(10))

    # Increase plot size to 10x8
    plt.figure(figsize=(10, 8))
    sns.heatmap(df_cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(f'Decision Tree Classifier\nAccuracy: {accuracy:.3f}')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

# Paths to the datasets
test_path = "/content/drive/My Drive/test_dataset.csv"
train_path = "/content/drive/My Drive/train_dataset.csv"

# Load and normalize the data
X_train, y_train = load_and_normalize_data(train_path)
X_test, y_test = load_and_normalize_data(test_path)

# Train and evaluate the Decision Tree model
train_and_evaluate_decision_tree(X_train, y_train, X_test, y_test)
```

Tabela 40: Código Matriz de Confusão Multi-Classe com DT.

4.7.1 *Output*

A matriz de confusão gerada pela árvore de decisão, Figura 33, segue o mesmo padrão da gerada anteriormente pelo K-NN. Ainda, é possível inferir que a taxa de aprendizado da DT é menor em comparação com o código anterior, uma vez que os parâmetros de avaliação, acurácia, precisão, *recall* e *F1 score*, em geral, diminuíram para cada classe, como pode ser visto abaixo.

1. **Modelo:** acurácia de 28,9%, precisão de 25,4%, *recall* de 28,9% e *F1 score* de 24,9%.
2. **Posição neutra:** acurácia de 66,8%, precisão de 27,1%, *recall* de 66,8% e *F1 score* de 38,6%.
3. **Extensão de pulso:** acurácia de 59%, precisão de 48,4%, *recall* de 59% e *F1 score* de 53,1%.
4. **Flexão de pulso:** acurácia de 61,3%, precisão de 46,7%, *recall* de 61,3% e *F1 score* de 53%.
5. **Desvio ulnar:** acurácia de 25,7%, precisão de 20,5%, *recall* de 25,7% e *F1 score* de 22,8%.
6. **Desvio radial:** acurácia de 18,1%, precisão de 24%, *recall* de 18,1% e *F1 score* de 20,6%.
7. **Punho fechado:** acurácia de 30,9%, precisão de 21,8%, *recall* de 30,9% e *F1 score* de 25,6%.
8. **Abdução dos dedos:** acurácia de 10,7%, precisão de 20,5%, *recall* de 10,7% e *F1 score* de 14%.
9. **Adução dos dedos:** acurácia de 3,1%, precisão de 14,8%, *recall* de 3,1% e *F1 score* de 5,1%.
10. **Supinação:** acurácia de 12,4%, precisão de 16,2%, *recall* de 12,4% e *F1 score* de 14,1%.
11. **Pronação:** acurácia de 1,2%, precisão de 13,9%, *recall* de 1,2% e *F1 score* de 2,2%.

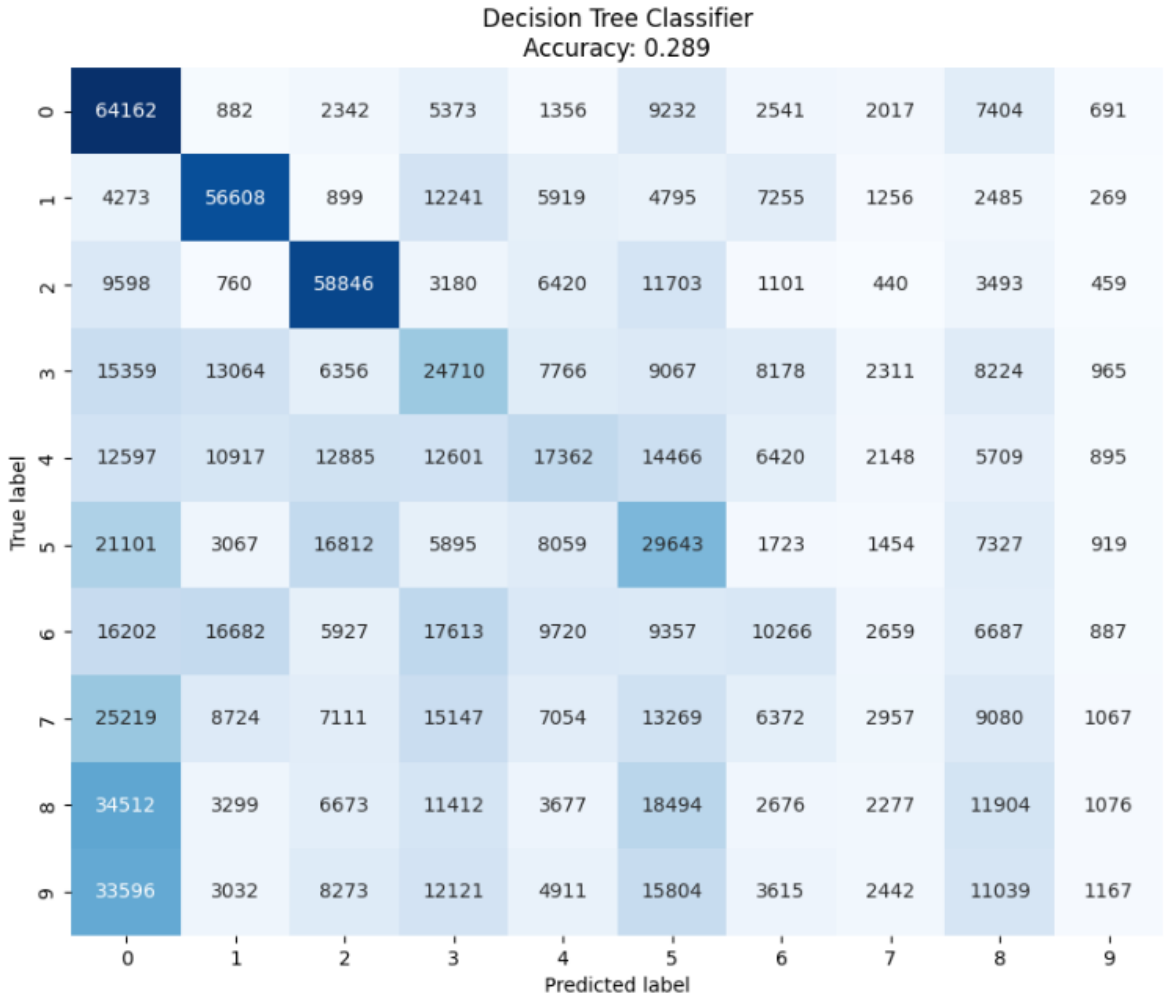

Figura 33: *Output* Matriz de Confusão Multi-Classe com DT.

4.8 K-means para Análise das Classes

Nesta seção, utilizou-se o algoritmo k-means para avaliar a similaridade entre as classes, analisando como os dados se agrupam e identificando possíveis padrões ou sobreposições entre elas. Essa abordagem permite entender melhor o comportamento dos dados e a proximidade entre diferentes classes, contribuindo para uma avaliação mais precisa da separabilidade entre os grupos.

```
1
2 # Import necessary libraries
3 import os
4 import numpy as np
5 import pandas as pd
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.cluster import KMeans
8 from sklearn.metrics import pairwise_distances
```

```
import matplotlib.pyplot as plt

# Load and normalize data
def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df.iloc[:, -1].values.astype(np.int32)  # Assuming last column has class labels (0-9)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels

# Perform K-Means clustering for a specified number of clusters
def perform_kmeans(data, n_clusters=10):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(data)
    return kmeans

# Calculate distances between cluster centroids
def calculate_centroid_distances(kmeans):
    centroids = kmeans.cluster_centers_
    distances = pairwise_distances(centroids)
    return distances

# Find similar classes based on centroid distances
def find_similar_classes(distances, threshold=1.0):
    n_clusters = distances.shape[0]
    similar_pairs = []

    for i in range(n_clusters):
        for j in range(i + 1, n_clusters):
            if distances[i, j] < threshold:
                similar_pairs.append((i, j, distances[i, j]))

    return similar_pairs

# Visualize the centroid distances as a heatmap
def plot_centroid_distances(distances):
    plt.figure(figsize=(8, 6))
    plt.imshow(distances, interpolation='nearest', cmap='Blues')
```

```
48      plt.colorbar()
49      plt.title('Distances Between Cluster Centroids')
50      plt.xlabel('Cluster Index')
51      plt.ylabel('Cluster Index')
52      plt.show()
53
54  # Define the file path for your data
55  data_path = "/content/drive/My Drive/random_output.csv"
56
57  # Check if the file path exists
58  if not os.path.exists(data_path):
59      raise FileNotFoundError(f"Data path {data_path} does not exist.")
60
61  # Load and normalize the data
62  data, labels = load_and_normalize_data(data_path)
63
64  # Perform K-Means clustering (number of clusters can be 10 for classes 0-9)
65  kmeans = perform_kmeans(data, n_clusters=10)
66
67  # Calculate and plot centroid distances
68  centroid_distances = calculate_centroid_distances(kmeans)
69  plot_centroid_distances(centroid_distances)
70
71  # Find and print similar classes
72  similar_classes = find_similar_classes(centroid_distances, threshold=2.5)
      # Adjust threshold as needed
73  print("Similar class pairs (based on centroid distances):")
74  for pair in similar_classes:
75      print(f"Class {pair[0]} and Class {pair[1]} are similar with a distance
      of {pair[2]:.2f}")
```

Tabela 41: Código de K-means para Análise das Classes.

4.8.1 *Output*

Na Figura 34, quanto mais claro for, mais próxima as classes são. Como pode ser averiguado, as classes 6 e 8 podem ser consideradas iguais do ponto de vista de divisão dos dados, com uma distância de 0,82, além disso, as mesmas são próximas das demais classes.

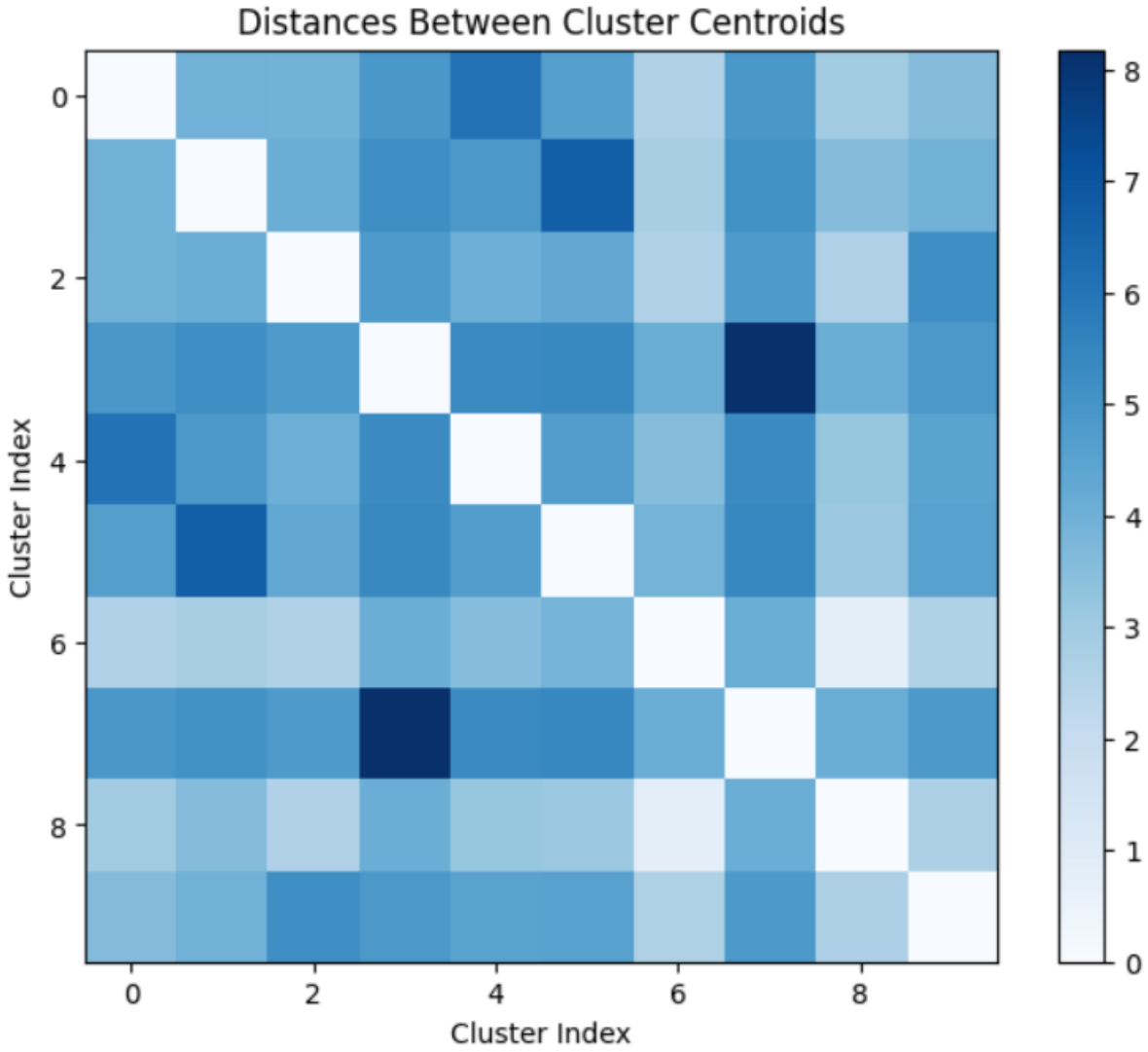

Figura 34: Distância entre os centroides dos *clusters*.

4.9 Análises Secundárias

4.9.1 Naive Bayes

O Naive Bayes é um algoritmo de classificação baseado no Teorema de Bayes, que é um princípio fundamental na teoria das probabilidades. O termo "Naive"(ingênuo) se refere ao fato de que o algoritmo faz uma suposição simplificadora: ele assume que as características (ou atributos) do conjunto de dados são independentes entre si, dado o valor da classe. Na prática, isso significa que o algoritmo assume que a presença ou ausência de uma característica particular não está relacionada à presença ou ausência de qualquer outra característica, o que nem sempre é verdadeiro. Apesar dessa suposição simplificadora, o Naive Bayes funciona surpreendentemente bem em muitos problemas do mundo real.

Como Funciona o Naive Bayes

1 - Teorema de Naive Bayes - O teorema de Bayes é a base do algoritmo e é usado para calcular a probabilidade posterior de uma classe, dado um conjunto de características. Ele pode ser expresso como:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \tag{4.4}$$

Onde:

- $P(C|X)$: Probabilidade posterior da classe C dado o vetor de características X;
- $P(X|C)$: Probabilidade de observar X dado que a classe é C;
- $P(C)$: Probabilidade a priori da classe C;
- $P(X)$: Probabilidade a priori do vetor de características X.

2 - Independência Ingênua - O Naive Bayes assume que todas as características são independentes entre si. Isso permite que o cálculo da probabilidade $P(X|C)$ seja simplificado como o produto das probabilidades individuais de cada característica:

$$P(X|C) = P(x_1|C) \cdot P(x_2|C) \cdot ... \cdot P(x_n|C) \tag{4.5}$$

3 - Classificação - Para classificar um novo exemplo, o algoritmo calcula a probabilidade de cada classe, dado o conjunto de características do exemplo, e escolhe a classe com a maior probabilidade.

Tipos de Naive Bayes

Existem várias versões do Naive Bayes, dependendo do tipo de dados:

- Gaussian Naive Bayes: Usado quando as características contínuas seguem uma distribuição normal (gaussiana). É o mais utilizado em dados que podem ser aproximados por uma distribuição normal;
- Multinomial Naive Bayes: Adequado para dados discretos, como contagem de palavras em um texto. Muito usado em problemas de classificação de textos;
- Bernoulli Naive Bayes: Usado para dados binários (0 ou 1), como a presença ou ausência de uma palavra em um documento.

Vantagens

- Simplicidade e Eficiência: O Naive Bayes é simples de implementar e computacionalmente eficiente, mesmo para grandes conjuntos de dados;
- Robustez com Poucos Dados: Funciona bem mesmo com um número relativamente pequeno de dados de treinamento;
- Bom Desempenho: Apesar da suposição de independência ingênua, o algoritmo geralmente tem um desempenho muito bom, especialmente em problemas de classificação de texto.

Desvantagens

- Independência de Características: A suposição de independência raramente é verdadeira na prática, o que pode limitar a precisão do modelo em alguns casos;
- Estimativa de Probabilidades: Se uma característica em um novo dado de entrada não foi observada durante o treinamento, a probabilidade correspondente pode ser zero, resultando em problemas de classificação. Isso é frequentemente tratado com técnicas como suavização de Laplace.

Comparação com Códigos Mais Complexos

```
from sklearn.naive_bayes import GaussianNB
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels
```

```
13
14 # Datasets
15 test_path = "/content/drive/My Drive/test_dataset.csv"
16 train_path = "/content/drive/My Drive/train_dataset.csv"
17
18 # Load and normalize data
19 X_train, y_train = load_and_normalize_data(train_path)
20 X_test, y_test = load_and_normalize_data(test_path)
21
22 # Naive Bayes classifier
23 nb_classifier = GaussianNB().fit(X_train, y_train)
24
25 # Predictions
26 y_nb_predictions = nb_classifier.predict(X_test)
27
28 # Output predictions and accuracy
29 print(y_nb_predictions)
30 print(nb_classifier.score(X_test, y_test))
```

Tabela 42: Comparação com Códigos Mais Complexos (Naive Bayes).

Ao comparar os resultados abaixo com os anteriores, nota-se uma piora significativa no desempenho do modelo. Além disso, quando a IA não consegue identificar a classe correta, ela tende a classificar esses casos como pertencentes à classe 0, o que indica uma inclinação para fazer suposições para essa categoria. Esse comportamento reflete uma falta de aprendizado efetivo nas classes mais difíceis de diferenciar, resultando em uma maior taxa de falsos positivos para a classe 0.

1. **Modelo:** acurácia de 24,9%, precisão de 27,2%, *recall* de 24,9% e *F1 score* de 22,4%.
2. **Posição neutra:** acurácia de 84,4%, precisão de 18,7%, *recall* de 84,4% e *F1 score* de 30,6%.
3. **Extensão de pulso:** acurácia de 49,2%, precisão de 59,2%, *recall* de 84,4% e *F1 score* de 30,6%.
4. **Flexão de pulso:** acurácia de 39%, precisão de 62,3%, *recall* de 39% e *F1 score* de 47,9%.
5. **Desvio ulnar:** acurácia de 19,5%, precisão de 21,8%, *recall* de 39% e *F1 score* de

47,9%.

6. **Desvio radial:** acurácia de 10%, precisão de 30,6%, *recall* de 10% e *F1 score* de 15,1%.
7. **Punho fechado:** acurácia de 20,5%, precisão de 22,3%, *recall* de 20,5% e *F1 score* de 21,4%.
8. **Abdução dos dedos:** acurácia de 7,5%, precisão de 21%, *recall* de 7,5% e *F1 score* de 11,1%.
9. **Adução dos dedos:** acurácia de 3,6%, precisão de 12,7%, *recall* de 3,6% e *F1 score* de 5,6%.
10. **Supinação:** acurácia de 11,8%, precisão de 13,4%, *recall* de 11,8% e *F1 score* de 12,6%.
11. **Pronação:** acurácia de 3,8%, precisão de 10,5%, *recall* de 3,8% e *F1 score* de 5,6%.

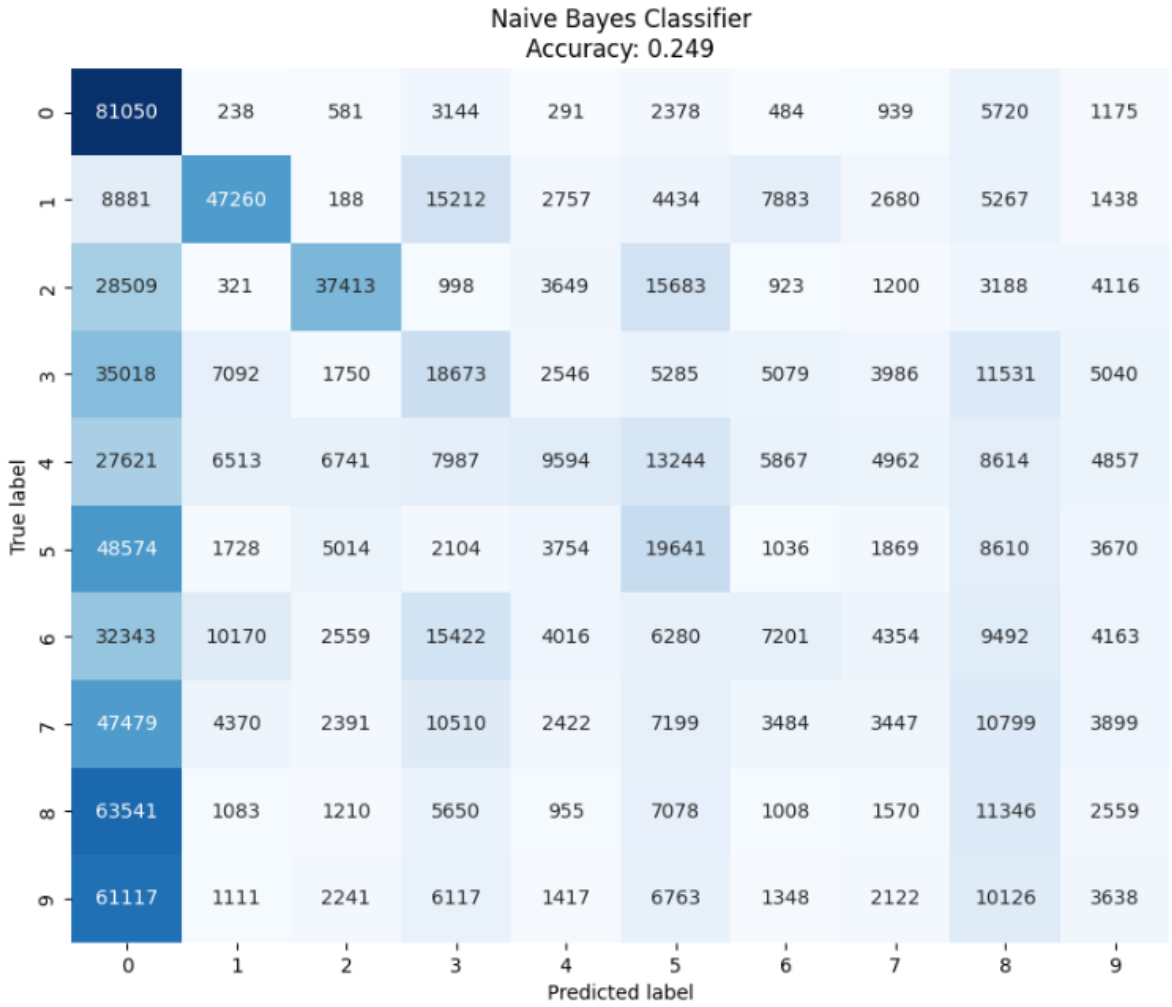

Figura 35: Acurácia Naive Bayes Classifier.

4.9.2 Gradient Boost Decision Trees

O modelo apresentou um alto custo computacional, levando aproximadamente 5 horas para ser executado, e resultou em uma acurácia de 0,29695, similar ao desempenho anterior. Esse tempo de processamento elevado, aliado à melhora mínima na acurácia, indica que o aumento na complexidade do modelo não trouxe benefícios significativos em termos de precisão, tornando o custo computacional desproporcional aos ganhos obtidos.

```
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels

# Datasets
test_path = "/content/drive/My Drive/test_dataset.csv"
train_path = "/content/drive/My Drive/train_dataset.csv"

# Load and normalize data
X_train, y_train = load_and_normalize_data(train_path)
X_test, y_test = load_and_normalize_data(test_path)

# Gradient Boosted Decision Trees classifier
gb_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate
    =0.1, random_state=42)
gb_classifier.fit(X_train, y_train)

# Predictions
y_gb_predictions = gb_classifier.predict(X_test)

# Output predictions and accuracy
print(y_gb_predictions)
print(gb_classifier.score(X_test, y_test))
```

Tabela 43: Código Gradient Boost Decision Trees.

4.9.3 Random Forest

O modelo apresentou um elevado custo computacional devido à complexidade e ao grande volume dos dados. Esse processamento exigiu tanto tempo que não foi possível executá-lo no ambiente do Colab, pois o tempo estimado ultrapassou 6 horas.

```
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float32)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
    return data, labels

# Datasets
test_path = "/content/drive/My Drive/test_dataset.csv"
train_path = "/content/drive/My Drive/train_dataset.csv"

# Load and normalize data
X_train, y_train = load_and_normalize_data(train_path)
X_test, y_test = load_and_normalize_data(test_path)

# Gradient Boosted Decision Trees classifier
gb_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate
    =0.1, random_state=42)
gb_classifier.fit(X_train, y_train)

# Predictions
y_gb_predictions = gb_classifier.predict(X_test)

# Output predictions and accuracy
print(y_gb_predictions)
```

```
31 print(gb_classifier.score(X_test, y_test))
```

Tabela 44: Código Random Forest.

5 Comparação da Análise com os Movimentos

Analisando os gráficos de matriz de confusão, como exemplificado na Figura 31, foi possível identificar alguns padrões. Primeiramente, as três primeiras classes apresentam uma alta taxa de acerto e uma grande diferenciação entre si. Por outro lado, as demais classes tendem a ser confundidas com a Classe 0, sendo que as duas últimas classes exibem a maior taxa de confusão. Além disso, a Classe 4 demonstra a maior distribuição de confusão, apresentando uma quantidade similar de tentativas errôneas em todas as classes. Assim, com base nesse tipo de gráfico, pode-se inferir que apenas as três primeiras classes têm um impacto positivo no aprendizado de máquina, enquanto as demais fornecem dados confusos.

Ademais, a análise das distâncias entre os *clusters* revelou que as Classes 6 e 8, correspondentes à abdução dos dedos e à supinação, podem ser consideradas equivalentes do ponto de vista do aprendizado de máquina.

Portanto, os movimentos de posição neutra, flexão, e extensão do pulso são classes distinguíveis a partir dos dados fornecidos. Em contraste, os outros movimentos geram sinais EMG semelhantes à posição neutra. Além disso, alguns desses movimentos, por serem contínuos, estão representados por dados discretos, como discutido na Seção 2, o que dificulta a separação clara em *clusters*.

6 Decisões Sobre Quais Movimentos e Eletrodos

Dado o estudo acerca dos resultados apresentados anteriormente, decidiu-se filtrar os dados e verificar os resultados. Primeiramente, como mostrado na Figura 25, foram mantidos apenas os dois sensores mais relevantes na diferenciação dos dados, 4 e 2. Após isso, os dados foram filtrados para análise sem o movimento de abdução de dedos, as seis primeiras classes e as três mais importantes para o *machine learning*. Para as análises posteriores, utilizou-se a avaliação dos gráficos de K-NN e K-means para proximidade dos *clusters*.

6.1 Diminuição dos sensores de entrada

Os resultados obtidos pela retirada dos sensores 1 e 3 não foram satisfatórios. Com o gráfico da Figura 36 foi possível observar que a retirada desses influiu em uma piora na capacidade de diferenciação em comparado com 24.

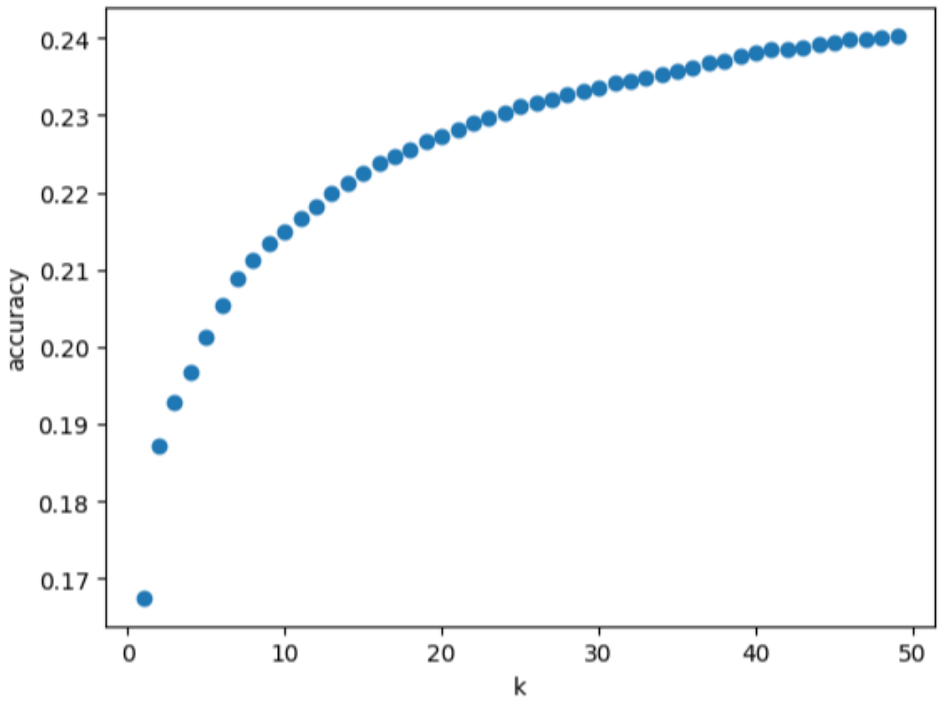

Figura 36: Acurácia do K-NN sem os sensores 1 e 3

A avaliação pelo K-means, Figura 37, mostrou que a distância entre os *clusters* diminuiu, sendo a menor 0,7, o que indica uma maior confusão entre as classes. Assim, o uso de apenas dois sensores levou a uma piora na classificação dos dados, sendo necessária a manutenção de todos os sensores, mesmo que alguns contribuam pouco para o resultado final.

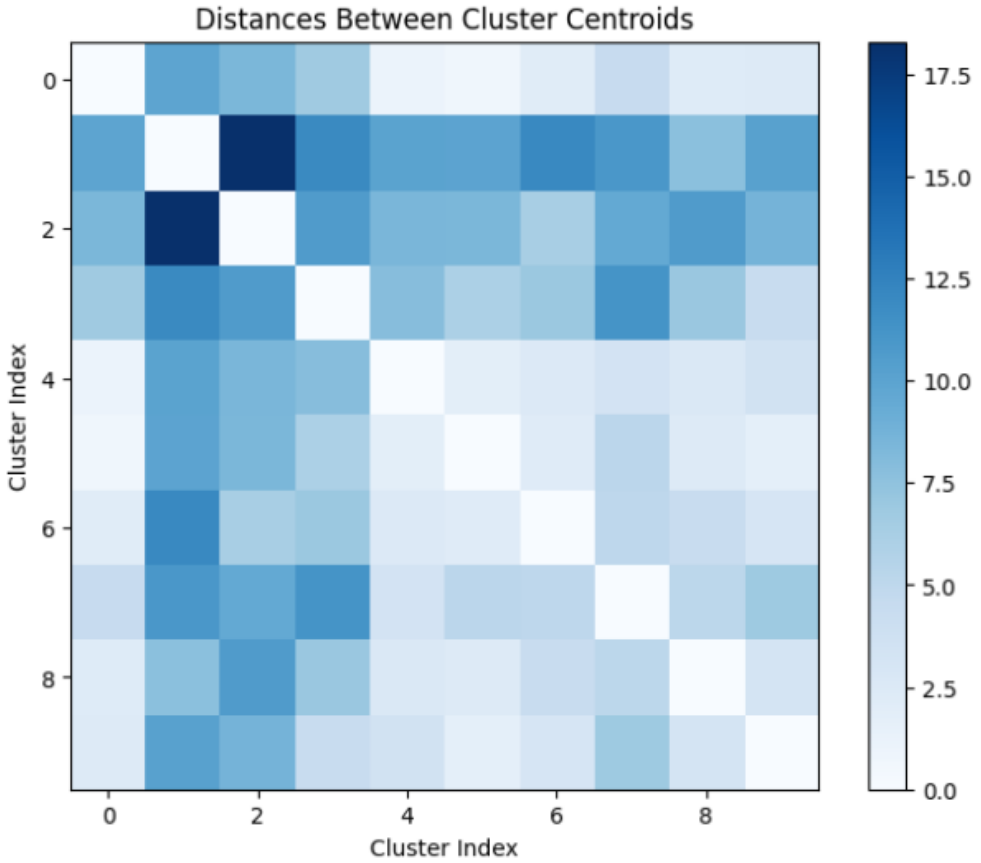

Figura 37: Distância entre *clusters* sem os sensores 1 e 3

6.2 Filtragem de classes

6.2.1 Retirada da classe 6

Nesta primeira avaliação, a Classe 6 foi filtrada devido à sua confusão com a Classe 8. Observou-se uma melhoria na separação dos movimentos, pois a acurácia exibida pelo K-NN, Figura 38, aumentou, e houve uma melhoria na distância entre as classes, Figura 39, com a menor distância sendo 1,94.

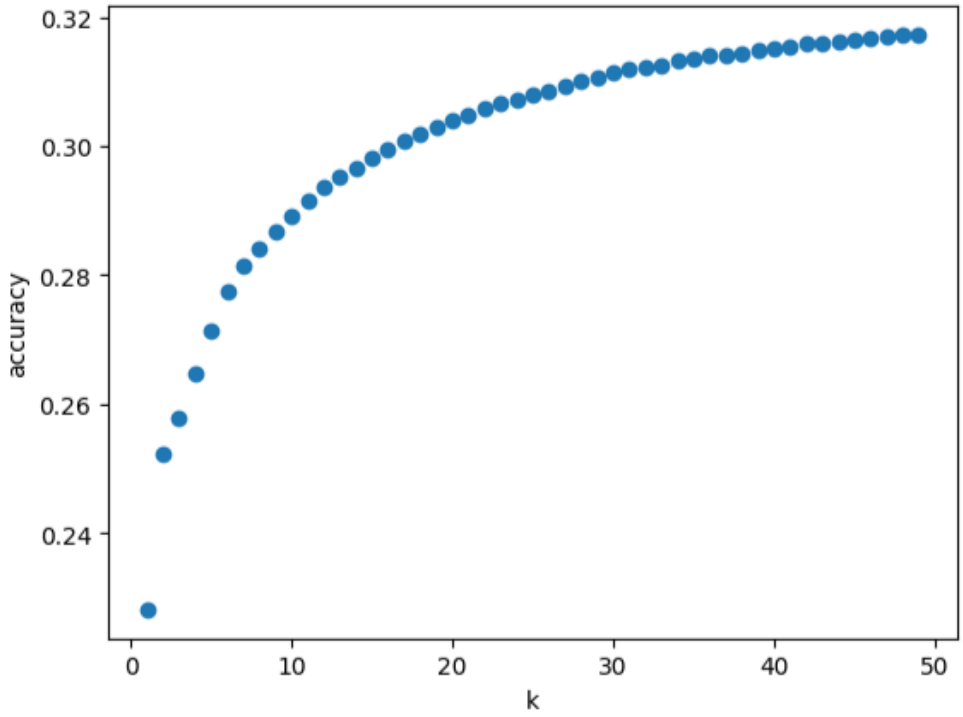

Figura 38: Acurácia do K-NN sem a classe 6

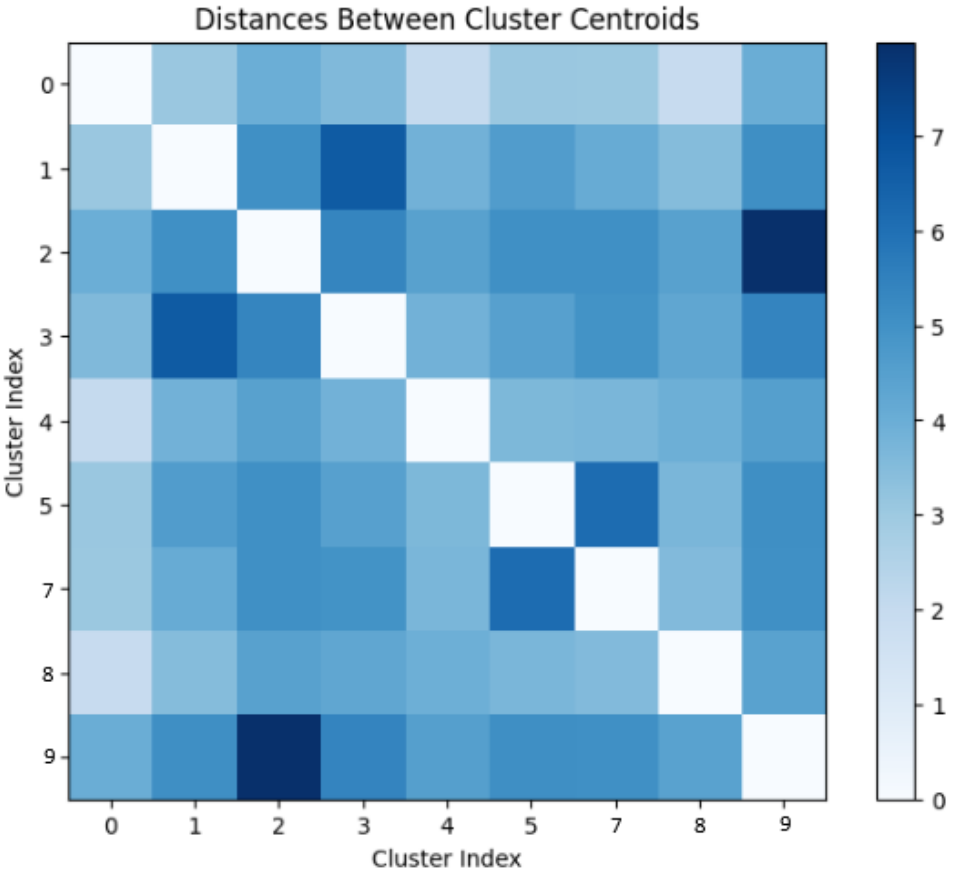

Figura 39: Distância entre *clusters* sem a classe 6

6.2.2 Somente as 6 primeiras classes

Com base no resultado anterior, considerou-se vantajoso filtrar os movimentos com menor precisão. Assim, foram removidas as quatro últimas classes, que representam movimentos contínuos, mantendo apenas as estacionárias.

Como antes, analisou-se a acurácia do K-NN, Figura 40, alcançando 48% de acerto, aproximadamente três vezes maior que um chute aleatório. A avaliação das distâncias no K-means, Figura 41, mostrou pouca melhoria, com a menor distância sendo 2,05.

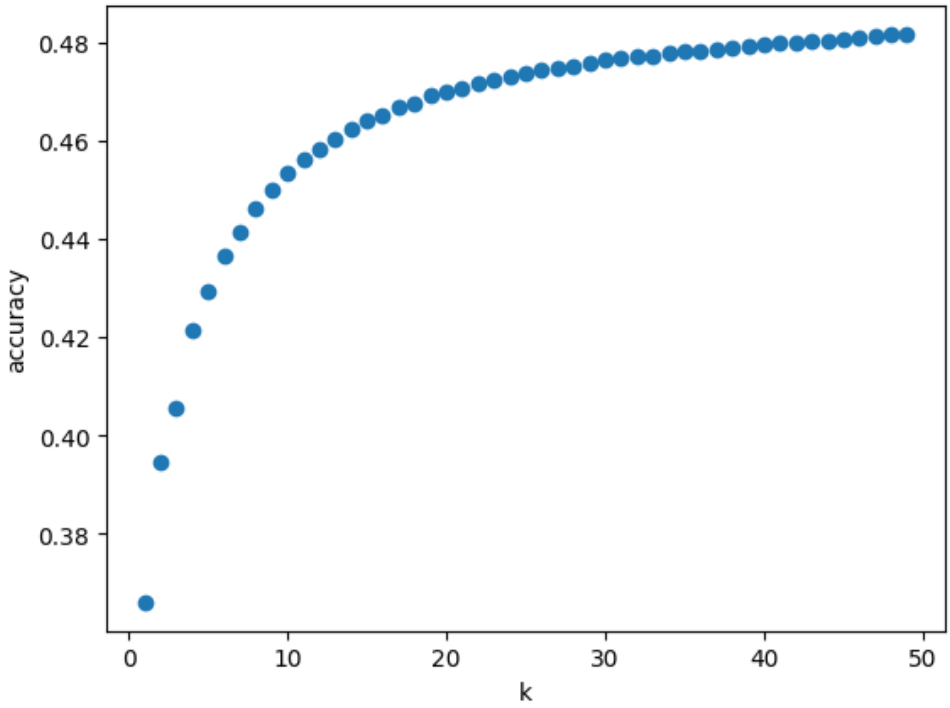

Figura 40: Acurácia do K-NN somente das 6 primeiras classes

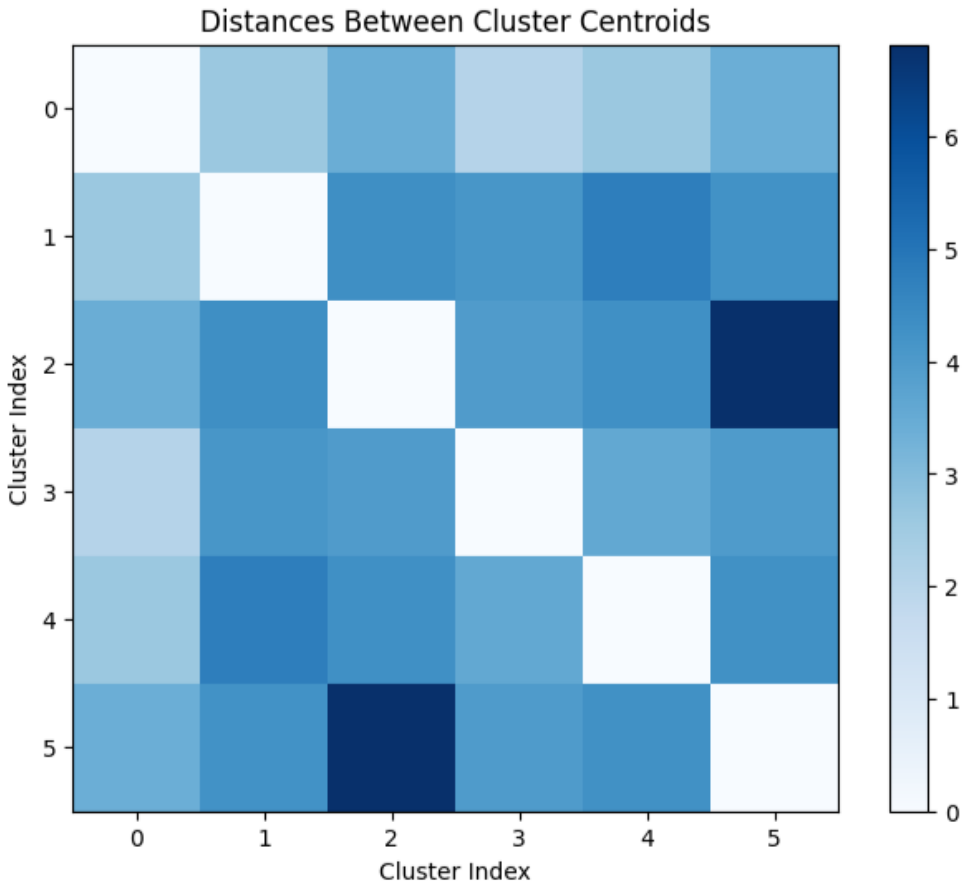

Figura 41: Distância entre *clusters* somente das 6 primeiras classes

Com esses dados, analisou-se a matriz de confusão do K-NN, Figura 42, focando na acurácia individual de cada classe, resultando nos seguintes valores::

1. **Posição neutra:** acurácia de 70,6%, precisão de 47,5%, *recall* de 70,6% e *F1 score* de 56,8%.

2. **Extensão de pulso:** acurácia de 64,6%, precisão de 60,8%, *recall* de 64,6% e *F1 score* de 62,7%.

3. **Flexão de pulso:** acurácia de 63,1%, precisão de 58,6%, *recall* de 63,1% e *F1 score* de 60,8%.

4. **Desvio ulnar:** acurácia de 32%, precisão de 35,1%, *recall* de 32% e *F1 score* de 33,5%.

5. **Desvio radial:** acurácia de 22,4%, precisão de 33,1%, *recall* de 22,4% e *F1 score* de 26,7%.

6. **Punho fechado:** acurácia de 29,3%, precisão de 37,2%, *recall* de 29,3% e *F1 score* de 32,8%.

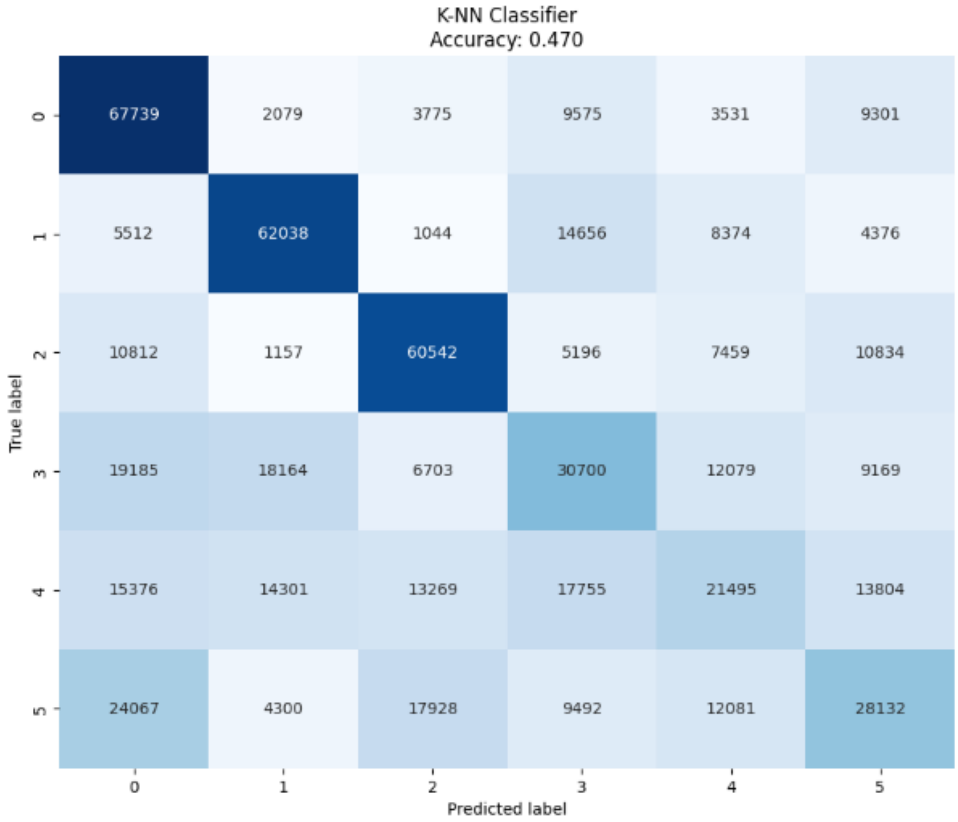

Figura 42: Matriz de confusão somente das 6 primeiras classes

Portanto, percebeu-se que a Classe 4 (desvio radial) continua a confundir a rede neural e que os três primeiros movimentos possuem maior capacidade de diferenciação. Assim, optou-se por manter apenas os dados EMG referentes aos movimentos com maior influência positiva no treinamento da IA.

6.2.3 3 classes mais importantes

Utilizando o código referente ao K-NN, construiu-se os gráficos de acurácia e de matriz de confusão, Figuras 43 e 44, respectivamente.

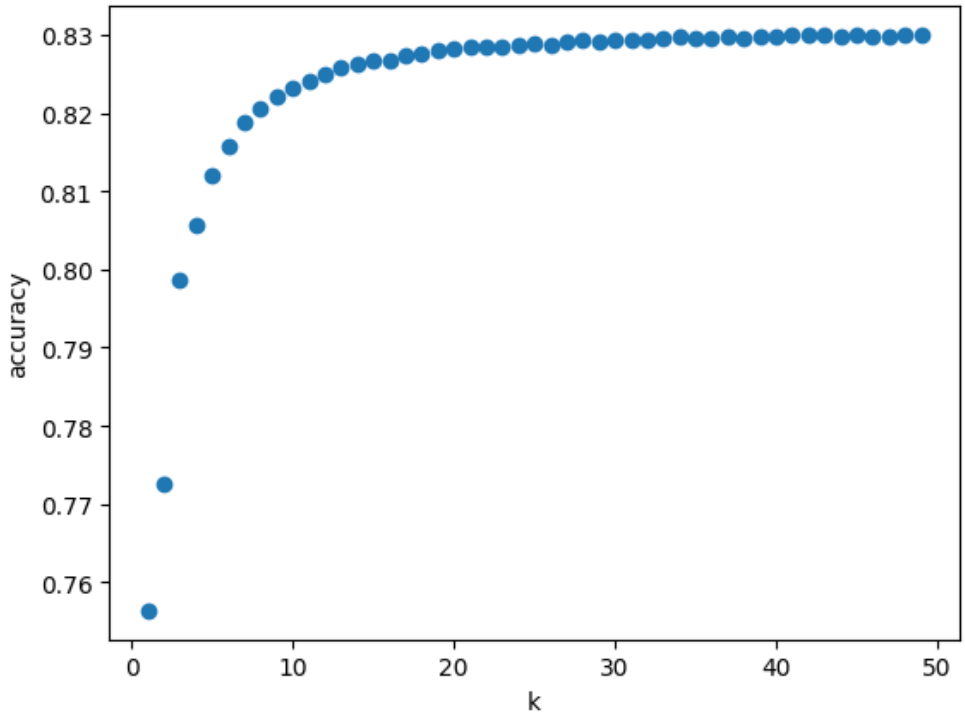

Figura 43: Acurácia do K-NN referente aos 3 primeiros movimentos

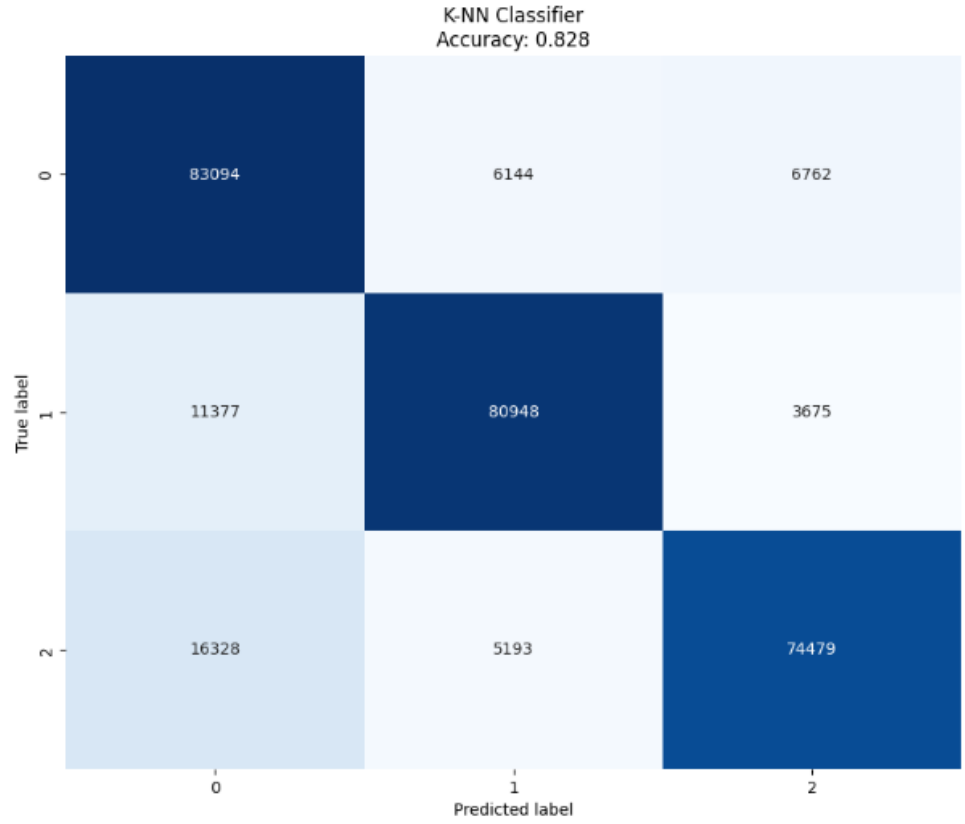

Figura 44: Matriz de confusão referente aos 3 primeiros movimentos.

Com eles foi possível chegar em uma taxa alta de acertos, 82,8%, e, ao aprofundar em cada movimento tem-se:

1. **Posição neutra:** acurácia de 86,6%, precisão de 75%, *recall* de 86,6% e *F1 score* de 80,4%.

2. **Extensão de pulso:** acurácia de 84,3%, precisão de 87,7%, *recall* de 84,3% e *F1 score* de 86%.

3. **Flexão de pulso:** acurácia de 77,6%, precisão de 87,7%, *recall* de 77,6% e *F1 score* de 82,3%.

Desse modo, indicando que a Classe 1 apresenta o melhor desempenho geral, devido à sua alta precisão, *recall* e *F1 Score*. Já a Classe 0, embora tenha uma acurácia elevada, apresenta uma precisão menor, sugerindo que é mais comumente referenciada, possivelmente com maior número de falsos positivos. Por outro lado, a Classe 2 apresenta o menor desempenho em termos de acurácia e *recall*, o que sugere que o modelo está deixando de identificar corretamente muitos exemplos reais dessa classe, indicando uma menor aprendizagem sobre ela.

7 MLP Final

7.1 Código

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import pandas as pd
from sklearn.preprocessing import StandardScaler
from collections import Counter
from sklearn.metrics import confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

def load_and_normalize_data(file_path):
    df = pd.read_csv(file_path)
    data = df.iloc[:, :-1].values.astype(np.float64)
    labels = df['class'].values.astype(np.int64)
    scaler = StandardScaler()
    data = scaler.fit_transform(data)
```

```
18      return data, labels
19
20  def create_mlp_model(input_size, output_size, activation='relu',
        dropout_rate=0.3):
21      model = models.Sequential()
22      model.add(layers.InputLayer(input_shape=(input_size,)))
23      model.add(layers.Dense(1000, activation=activation))
24      model.add(layers.Dropout(dropout_rate))
25      model.add(layers.Dense(1200, activation=activation))
26      model.add(layers.Dropout(dropout_rate))
27      model.add(layers.Dense(1000, activation=activation))
28      model.add(layers.Dropout(dropout_rate))
29      model.add(layers.Dense(output_size, activation='softmax'))
30      return model
31
32  # Custom callback to compute and display confusion matrix after each epoch
33  class ConfusionMatrixCallback(tf.keras.callbacks.Callback):
34      def __init__(self, validation_data):
35          super().__init__()
36          self.validation_data = validation_data
37
38      def on_epoch_end(self, epoch, logs=None):
39          val_data, val_labels = [], []
40          for batch in self.validation_data:
41              data, labels = batch
42              val_data.append(data)
43              val_labels.append(labels)
44          val_data = np.concatenate(val_data, axis=0)
45          val_labels = np.concatenate(val_labels, axis=0)
46          predictions = np.argmax(self.model.predict(val_data), axis=1)
47          conf_matrix = confusion_matrix(val_labels, predictions)
48          accuracy = accuracy_score(val_labels, predictions)
49          precision = precision_score(val_labels, predictions, average='macro
        ')
50          recall = recall_score(val_labels, predictions, average='macro')
51          f1 = f1_score(val_labels, predictions, average='macro')
52
53          print(f"\nEpoch {epoch + 1}:")
54          print(f"Confusion Matrix:\n{conf_matrix}")
```

```
55          print(f"Acur cia: {accuracy:.4f}, Precis o: {precision:.4f},
    Recall: {recall:.4f}, F1 Score: {f1:.4f}")
56
57          plt.figure(figsize=(8, 6))
58          sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=
    False)
59          plt.title(f'Confusion Matrix - Epoch {epoch + 1}')
60          plt.ylabel('True label')
61          plt.xlabel('Predicted label')
62          plt.show()
63
64 # Custom callback to plot loss and val_loss at each epoch
65 class LossHistoryPlotCallback(tf.keras.callbacks.Callback):
66      def _init_(self):
67          super()._init_()
68          self.losses = []
69          self.val_losses = []
70
71      def on_epoch_end(self, epoch, logs=None):
72          self.losses.append(logs.get('loss'))
73          self.val_losses.append(logs.get('val_loss'))
74          epochs_range = range(1, epoch + 2)
75          plt.figure(figsize=(8, 6))
76          plt.plot(epochs_range, self.losses, label='Training Loss')
77          plt.plot(epochs_range, self.val_losses, label='Validation Loss')
78          plt.title(f'Loss per Epoch - Epoch {epoch + 1}')
79          plt.xlabel('Epochs')
80          plt.ylabel('Loss')
81          plt.legend()
82          plt.grid(True)
83          plt.show()
84
85 # Learning Rate Scheduler
86 def scheduler(epoch, lr):
87      if epoch < 10:
88          return lr
89      else:
90          return lr * 0.9  # Reduz 10% da taxa de aprendizado a cada  poca
    ap s a 10
91
```

```
92 lr_scheduler = tf.keras.callbacks.LearningRateScheduler(scheduler)
93
94 # Paths for data
95 data_path = "/content/drive/MyDrive/filtered_dataset/classes/
       filtered_file_wthout_class3456789.csv"
96 train_path = "/content/drive/MyDrive/filtered_dataset/classes/
       train_dataset_without_class3456789.csv"
97 val_path = "/content/drive/MyDrive/filtered_dataset/classes/
       val_dataset_without_class3456789.csv"
98 test_path = "/content/drive/MyDrive/filtered_dataset/classes/
       test_dataset_without_class3456789.csv"
99
100 input_size = 4  # Assuming 4-channel EMG data
101 output_size = 10  # 10 hand gestures
102 learning_rate = 0.00002
103 num_epochs = 10
104 batch_size = 20
105 dropout_rate = 0.3
106
107 for path in [data_path, train_path, val_path, test_path]:
108     if not os.path.exists(path):
109         raise FileNotFoundError(f"Data path {path} does not exist.")
110
111 data1, labels1 = load_and_normalize_data(data_path)
112 data2, labels2 = load_and_normalize_data(train_path)
113 data3, labels3 = load_and_normalize_data(val_path)
114 data4, labels4 = load_and_normalize_data(test_path)
115
116 # Counting classes
117 def count_labels(labels):
118     return Counter(labels)
119
120 print("Contagem de classes:", count_labels(labels1))
121 print("Contagem de classes:", count_labels(labels2))
122 print("Contagem de classes:", count_labels(labels3))
123 print("Contagem de classes:", count_labels(labels4))
124
125 train_dataset = tf.data.Dataset.from_tensor_slices((data2, labels2)).
       shuffle(len(data2), seed=42).batch(batch_size)
```

```
126 val_dataset = tf.data.Dataset.from_tensor_slices((data3, labels3)).shuffle(
    len(data3), seed=42).batch(batch_size)
127 test_dataset = tf.data.Dataset.from_tensor_slices((data4, labels4)).shuffle
    (len(data4), seed=42).batch(batch_size)
128
129 model = create_mlp_model(input_size, output_size, dropout_rate=dropout_rate
    )
130 model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=
    learning_rate),
131               loss='sparse_categorical_crossentropy',
132               metrics=['accuracy'])
133
134 model.summary()
135
136 # Callbacks
137 early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=10, restore_best_weights=True)
138 model_checkpoint = tf.keras.callbacks.ModelCheckpoint('best_model.keras',
    save_best_only=True)
139 confusion_matrix_callback = ConfusionMatrixCallback(validation_data=
    val_dataset)
140 csv_logger = tf.keras.callbacks.CSVLogger('training_log.csv')
141 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='./logs')
142 terminate_on_nan = tf.keras.callbacks.TerminateOnNaN()
143 reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor
    =0.5, patience=2, min_lr=1e-6, verbose=1)
144 loss_plot_callback = LossHistoryPlotCallback()
145
146 # Train with all the callbacks
147 history = model.fit(train_dataset, validation_data=val_dataset, epochs=
    num_epochs, verbose=1,
148                     callbacks=[early_stopping, model_checkpoint,
    backup_restore, lr_scheduler, confusion_matrix_callback,
149                                csv_logger, tensorboard_callback,
    terminate_on_nan, loss_plot_callback, reduce_lr])
150
151 # Test evaluation
152 model.load_weights('best_model.keras')
153 test_loss, test_accuracy = model.evaluate(test_dataset)
154 print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
155
156 # Manually compute confusion matrix for the test data
157 test_data, test_labels = [], []
158 for batch in test_dataset:
159     data, labels = batch
160     test_data.append(data)
161     test_labels.append(labels)
162
163 test_data = np.concatenate(test_data, axis=0)
164 test_labels = np.concatenate(test_labels, axis=0)
165
166 # Generate predictions
167 predictions = np.argmax(model.predict(test_data), axis=1)
168
169 # Calculate the confusion matrix and other metrics
170 conf_matrix = confusion_matrix(test_labels, predictions)
171 accuracy = accuracy_score(test_labels, predictions)
172 precision = precision_score(test_labels, predictions, average='macro')
173 recall = recall_score(test_labels, predictions, average='macro')
174 f1 = f1_score(test_labels, predictions, average='macro')
175
176 print(f"\nConfusion Matrix:\n{conf_matrix}")
177 print(f"Acur cia: {accuracy:.4f}, Precis o: {precision:.4f}, Recall: {
    recall:.4f}, F1 Score: {f1:.4f}")
178
179 # Plot the confusion matrix
180 plt.figure(figsize=(8, 6))
181 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
182 plt.title('Confusion Matrix - Test Set')
183 plt.ylabel('True label')
184 plt.xlabel('Predicted label')
185 plt.show()
```

Tabela 45: MLP Final.

7.1.1 Importações de Bibliotecas

O código começa importando várias bibliotecas já citadas necessárias para manipulação de dados.

7.1.2 Funções Principais

$load\_and\_normalize\_data(file\_path)$

Carrega um arquivo CSV contendo dados e normaliza as características usando `StandardScaler`. Retorna os dados normalizados e os rótulos.

$create\_mlp\_model(input\_size, output\_size, activation =' relu', dropout\_rate = 0.3)$

Cria um modelo de rede neural multicamada (MLP) com várias camadas densas e camadas de dropout para prevenir overfitting. O modelo usa a função de ativação ReLU e termina com uma camada softmax para classificação.

ConfusionMatrixCallback

Uma classe personalizada que estende `tf.keras.callbacks.Callback`. Ela calcula e exibe a matriz de confusão após cada época do treinamento, além de calcular métricas como acurácia, precisão, recall e F1 score.

LossHistoryPlotCallback

Outra classe personalizada que armazena e plota a perda (loss) do treinamento e validação ao final de cada época.

scheduler(epoch, lr)

Uma função para ajustar a taxa de aprendizado durante o treinamento, reduzindo-a em 10% após a décima época.

7.1.3 Carregamento dos Dados

O script define caminhos para os conjuntos de dados (treinamento, validação e teste) e carrega os dados usando a função `load_and_normalize_data`. Ele também conta as classes presentes em cada conjunto usando a função `count_labels`.

7.1.4 Preparação dos Dados

Os dados são convertidos em datasets do TensorFlow (`tf.data.Dataset`) para facilitar o treinamento em lotes (batches).

7.1.5 Criação e Compilação do Modelo

Um modelo MLP é criado com as dimensões especificadas. O modelo é compilado com o otimizador Adam, uma função de perda para classificação categórica esparsa (`sparse_categorical_crossentropy`) e métrica de acurácia.

7.1.6 Treinamento do Modelo

O modelo é treinado usando o método `fit`, com várias callbacks para monitorar o progresso, salvar o melhor modelo, plotar perdas, entre outros.

7.1.7 Avaliação do Modelo

Após o treinamento, o modelo carrega os melhores pesos salvos e é avaliado no conjunto de testes. A matriz de confusão é calculada novamente, junto com as métricas relevantes.

7.1.8 Visualização

Finalmente, a matriz de confusão é plotada usando Seaborn para facilitar a interpretação dos resultados.

7.1.9 Importante Destacar

- *Custom callback to plot loss and val_loss at each epoch* - Interessante para se observar o comportamento do gráfico da *loss*, evitando o*overfitting* ou *underfitting*. Quando os gráficos estabilizam, o aprendizado para;
- *Custom callback to compute and display confusion matrix after each epoch* - Possibilita acompanhar como se comporta cada classe;
- O *batch size* foi menor para exigir menor custo computacional;
- O *learning rate* foi menor para procurar um vale de resposta de forma mais suave;
- O número de épocas foi menor pois com os outros testes indicaram que a estabilização ocorre na época 10.

7.2 *Output*

7.2.1 Arquitetura

A Figura 45 apresenta as camadas e o número de parâmetros do modelo. Esses parâmetros representam a quantidade de interações que a IA realiza para chegar a uma resposta, refletindo a complexidade e a capacidade do modelo de capturar padrões nos dados.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1000)	5,000
dropout (Dropout)	(None, 1000)	0
dense_1 (Dense)	(None, 1200)	1,201,200
dropout_1 (Dropout)	(None, 1200)	0
dense_2 (Dense)	(None, 1000)	1,201,000
dropout_2 (Dropout)	(None, 1000)	0
dense_3 (Dense)	(None, 10)	10,010

Total params: 2,417,210 (9.22 MB)
Trainable params: 2,417,210 (9.22 MB)
Non-trainable params: 0 (0.00 B)

Figura 45: Arquitetura final da MLP.

7.2.2 Última época

Na décima época, o modelo de aprendizado de máquina apresentou os seguintes resultados:

Acuracy: 0.8355

Precisão: 0.8395

Recall: 0.8355

F1 Score: 0.8361

Ainda, a Figura 46 mostra que a quantidade de falsos positivos é significativamente baixa em comparação com o número de acertos, indicando uma boa performance do modelo em termos de classificação correta.

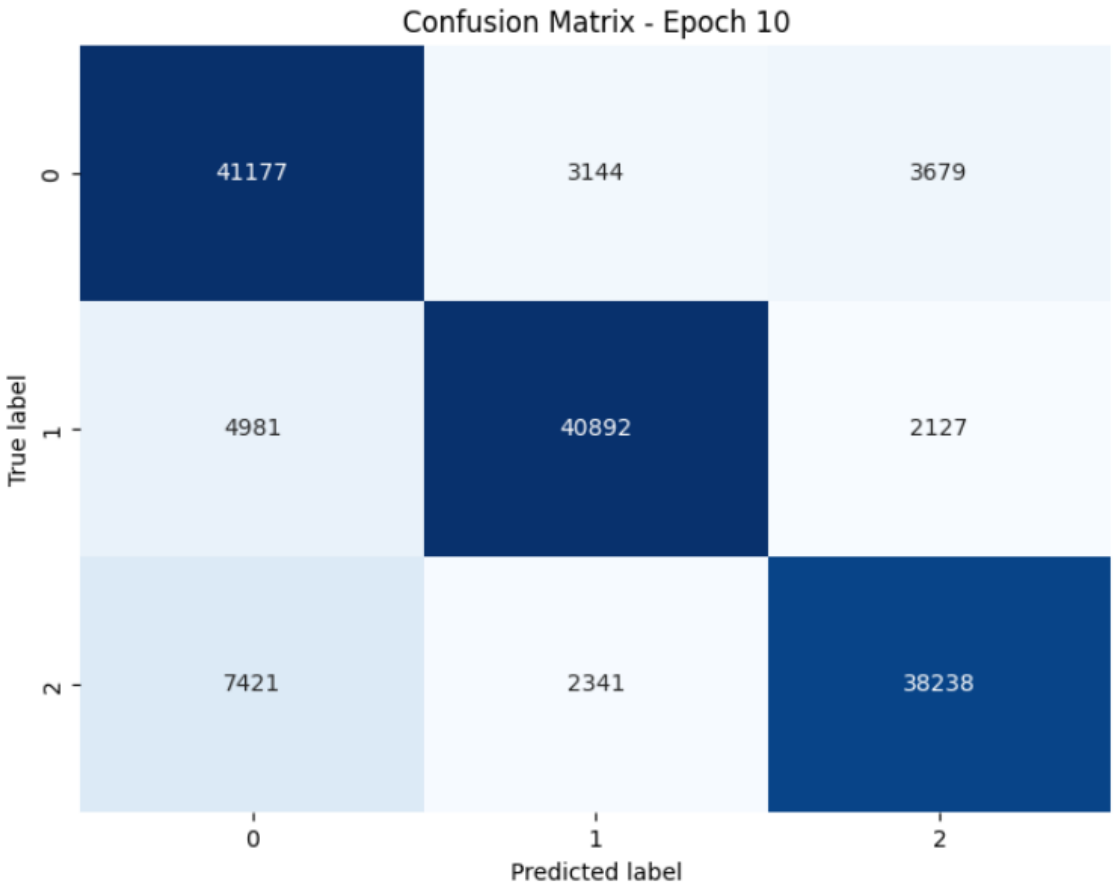

Figura 46: Matriz de confusão da época 10.

7.2.3 Gráfico de Perda

Ao construir um gráfico que mostra a evolução das *loss* de treinamento e validação, Figura 47, observa-se a estabilização da val_loss, o que indica que o modelo atingiu seu limite preditivo e deixou de melhorar seu desempenho, sugerindo que o aprendizado foi maximizado e não há mais progresso no processo de treinamento.

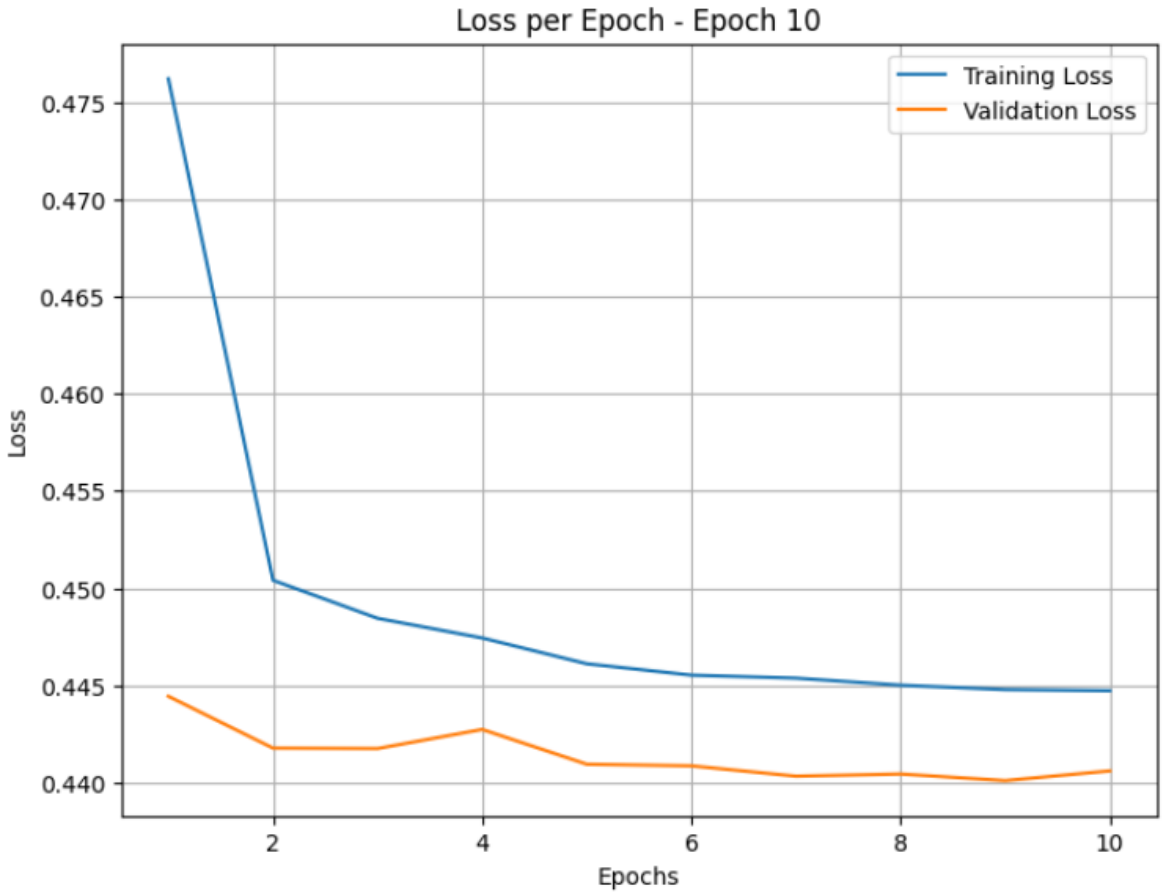

Figura 47: Gráfico da perda de todas as épocas.

7.2.4 Parâmetro da época 10

Nessa seção, são enunciados alguns dados relevantes da última época, acurácia de treino, *loss* do treino, acurácia da validação, *loss* da validação e *learning rate*:

$$50400/50400 \ \_\_\_\_\_ \ 2267s \ 45ms/step$$

$$\text{Acurácia: } 0.8329$$

$$Loss\text{: } 0.4449$$

$$val\_accuracy\text{: } 0.8355$$

$$val\_loss\text{: } 0.4406$$

$$learning\_rate\text{: } 1.0000e-05$$

Com esses dados percebe-se que no decorrer do treinamento foi necessário a utilização do *callback* de redução da *learning rate* e que cada época demorou em torno de 40 minutos para terminar.

7.2.5 Teste do modelo

Com o treinamento concluído de maneira satisfatória, avançou-se para a etapa final do modelo. Os resultados obtidos a partir do *test set* são apresentados a seguir:

Acurácia: 0.8336

Precisão: 0.8375

Recall: 0.8336

F1 Score: 0.8341

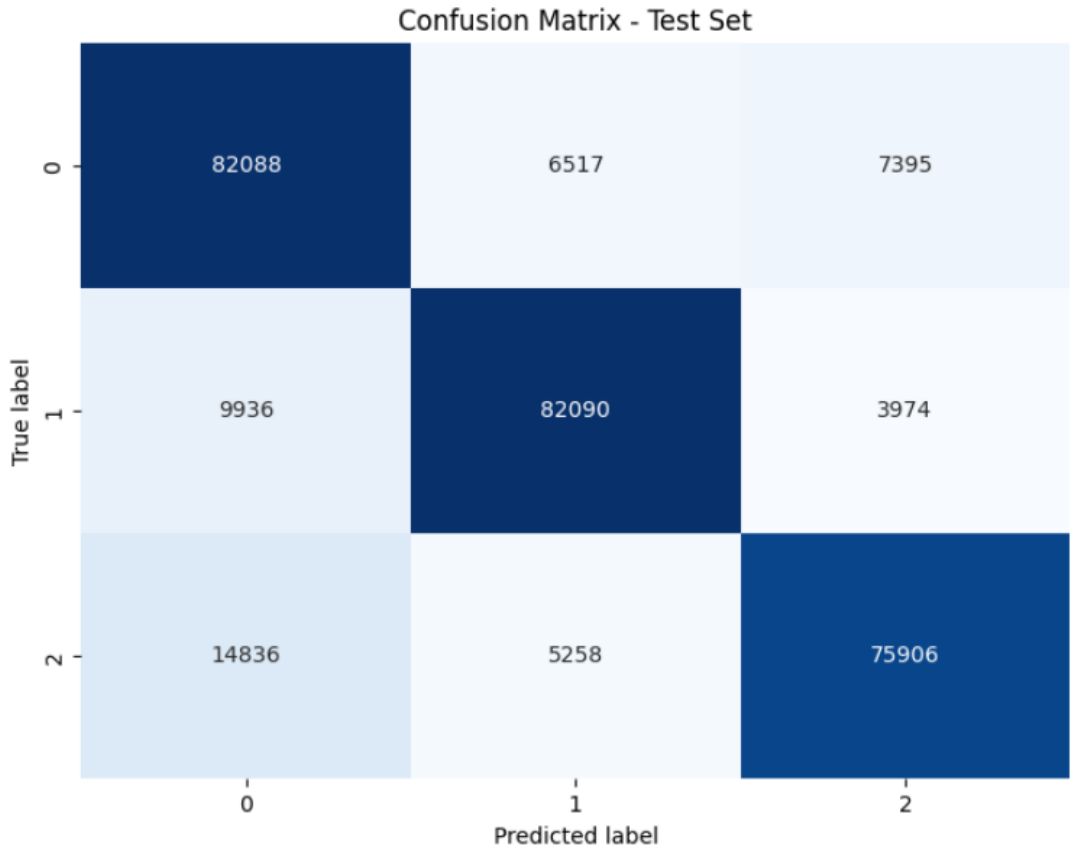

Figura 48: Dados da Matriz de Confusão Final do Teste.

Portanto, ao final, o modelo apresentou comportamento adequado no conjunto de teste. Com esses dados, pode-se afirmar que o modelo mostrou um desempenho equilibrado em relação às métricas de avaliação. A acurácia de 0.8336 indica que aproximadamente 83% das previsões do modelo foram corretas. A precisão de 0.8375 sugere que, das previsões feitas como positivas, 83.75% estavam corretas. O *recall* de 0.8336 indica que o modelo foi capaz de identificar corretamente 83.36% dos casos positivos reais. Por fim, o *F1 Score* de 0.8341 reflete o equilíbrio entre precisão e *recall*, sendo um valor que

indica que o modelo está performando de maneira eficaz, sem um grande viés para falsos positivos ou falsos negativos.

Esses resultados demonstram que a IA final possui uma boa capacidade de generalização e está bem ajustado para a classificação dos movimentos selecionados, com um desempenho consistente em todas as métricas importantes.

8 Conclusões

Durante a progressão deste trabalho encontrou-se imprevistos que demandaram maior atenção. Primeiramente, devido ao grande volume e complexidade dos dados, sua limpeza e organização demandaram um esforço considerável. Em um segundo momento, os testes de configurações variadas para otimizar o desempenho da IA implicaram em múltiplas iterações e ajustes, o que prolongou o tempo total dedicado à análise.

Com isso, supõe-se que os três primeiros movimentos analisados tiveram a maior taxa de acertos devido sua maior intensidade atrelada à contração muscular e ao posicionamento dos eletrodos. Além disso, outros movimentos tendem a utilizar grupamentos musculares semelhantes aos da posição neutra, o que pode gerar confusão na classificação. Portanto, o aprimoramento contínuo dos dados e do treinamento da IA é essencial para alcançar resultados ainda melhores no futuro.

Desse modo, como foi avaliado, a qualidade dos dados e do posicionamento dos eletrodos foram fatores críticos que impactaram o treinamento da inteligência artificial. Após um processo de filtragem e testes em diferentes configurações, foi alcançada uma acurácia elevada de 83,36%, permitindo a diferenciação entre a posição neutra, flexão e extensão do pulso. No entanto, ainda existem oportunidades para melhorias, especialmente na classe 2, que poderia ter seu *recall* aumentado, além de aprimorar a precisão da classe 0 para reduzir confusões com as outras classes. Este trabalho serve como base para futuros estudos sobre outras bases de dados, implementações de órteses e robôs anatômicos.

Referências

[1] Leila Maria Beltramini. *Elementos de histologia e anatomo-fisiologia humana*. IFSC, 1999.

[2] Charles F. Stevens. The neuron. *Scientific American*, 241(3):54–65, 1979.

[3] Arthur C Guyton and John E Hall. Textbook of medical physiology: With student consult online access (guyton physiology) by, 2005.

[4] Jana Vasković. Kenhub in... *Insula*, 1:10.

[5] Nei Augusto Andrade. Desenvolvimento de um sistema de aquisição e processamento de sinais eletromiográficos de superfície para a utilização no controle de próteses motoras ativas. 2007.

[6] MA Cavalcanti Garcia and TMM Vieira. Surface electromyography: Why, when and how to use it. *Revista andaluza de medicina del deporte*, 4(1):17–28, 2011.

[7] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9(1):381–386, 2020.

[8] Irwin B Levitan and Leonard K Kaczmarek. *The neuron: cell and molecular biology*. Oxford University Press, USA, 2015.

[9] Keiichiro Susuki. Myelin: a specialized membrane for cell communication. *Nature education*, 3(9):59, 2010.

[10] Alexander Kister and Ilya Kister. Overview of myelin, major myelin lipids, and myelin-associated proteins. *Frontiers in Chemistry*, 10:1041961, 2023.

[11] Richard L Lieber and Jan Fridén. Functional and clinical significance of skeletal muscle architecture. *Muscle & Nerve: Official Journal of the American Association of Electrodiagnostic Medicine*, 23(11):1647–1666, 2000.

[12] RMC BRANCALHÃO, LFC RIBEIRO, B LIMA, RI KUNZ, and MC CAVÉQUIA. Tecido muscular, 2016, 2018.

[13] Walter R Frontera and Julien Ochala. Skeletal muscle: a brief review of structure and function. *Calcified tissue international*, 96:183–195, 2015.

[14] Göran Lundborg and Birgitta Rosén. Hand function after nerve repair. *Acta physiologica*, 189(2):207–217, 2007.

[15] Brittney Mitchell and Lacey Whited. Anatomy, shoulder and upper limb, forearm muscles. In *StatPearls [Internet]*. StatPearls Publishing, 2023.

[16] James W Strickland. The scientific basis for advances in flexor tendon surgery. *Journal of Hand Therapy*, 18(2):94–110, 2005.

[17] DA Neumann. Elbow and forearm complex. *Kinesiology of the Musculoskeletal System: Foundations for Physical Rehabilitation*, pages 133–171, 2002.

[18] Terri M Skirven, A Lee Osterman, Jane Fedorczyk, and Peter C Amadio. *Rehabilitation of the hand and upper extremity, 2-volume set E-book: expert consult*. Elsevier Health Sciences, 2011.

[19] John G Kreifeldt and Sumner Yao. A signal-to-noise investigation of nonlinear electromyographic processors. *IEEE Transactions on Biomedical Engineering*, (4):298–308, 1974.

[20] CJ DeLuca. Motor units alive-understanding them one pulse at a time. In *Basmajian Lecture: Keynote address at the Proceedings of the XIIth Congress of the International Society of Electrophysiolgy and Kinesiology, Montreal*, page 2, 1998.

[21] A Aishath Murshida, BK Chaithra, B Nishmitha, PB Pallavi, S Raghavendra, and K Mahesh Prasanna. Survey on artificial intelligence. *Int J Comput Sci Eng*, 7:1778–1790, 2019.

[22] Mohammed Amine El Mrabet, Khalid El Makkaoui, and Ahmed Faize. Supervised machine learning: a survey. In *2021 4th International conference on advanced communication technologies and networking (CommNet)*, pages 1–10. IEEE, 2021.

[23] Teresa Bernarda Ludermir. Inteligência artificial e aprendizado de máquina: estado atual e tendências. *Estudos Avançados*, 35:85–94, 2021.

[24] Muhammad Ali Syakur, B Khusnul Khotimah, EMS Rochman, and Budi Dwi Satoto. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP conference series: materials science and engineering*, volume 336, page 012017. IOP Publishing, 2018.

[25] Rubens Correa Araujo. *Utilização da eletromiografia em análise biomecânica do movimento humano*. PhD thesis, Universidade de São Paulo, 1998.

[26] Basakuau Nkomi Nkosi Junior. A eletromiografia associada à inteligência artificial no diagnóstico de doenças e no rendimento físico. 2021.

[27] Paulo L Viana, Victoria S Fujii, Larissa M Lima, Gabriel L Ouriques, Gustavo Casagrande de Oliveira, Renato Varoto, and Alberto Cliquet Jr. An artificial neural network for hand movement classification using surface electromyography. In *BIOSIGNALS*, pages 185–192, 2019.

[28] Paulo Henrique Gomes Machado. Classificação de gestos das mãos usando plataformas vestíveis baseadas em eletromiografia de superfície no antebraço e unidades inerciais. 2018.

[29] Zsolt Lászlo Kovács, O Cérebro, and Sua Mente. uma introdução à neurociência computacional. *Edição Acadêmica, São Paulo*, 1997.

[30] Carroll E Cross. Bloom and fawcett: A textbook of histology. *JAMA*, 274(4):352–352, 1995.

[31] Rafael Lourenço do Carmo. Kenhub em... *Flexores superficiais e intermediários do antebraço*, 1:5.