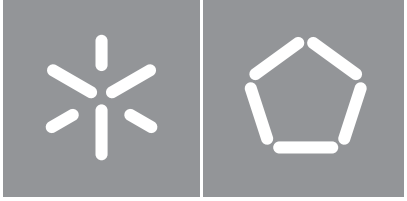
Universidade do Minho
Escola de Engenharia

Pedro Almeida Fernandes

Reconhecimento de atividades em *smartphones* usando sensores não intrusivos

Outubro, 2023



Universidade do Minho
Escola de Engenharia

Pedro Almeida Fernandes

Reconhecimento de atividades em *smartphones* usando sensores não intrusivos

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação de
Cesar Analide Freitas Silva Costa Rodrigues
Bruno Filipe Martins Fernandes

Outubro, 2023

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

_____, _____

(Localização) (Data)

_____

(Pedro Almeida Fernandes)

Agradecimentos

Gostaria de expressar a minha profunda gratidão a todas as pessoas que tornaram possível a realização deste trabalho. Em primeiro lugar, aos meus orientadores, Bruno Fernandes e Cesar Analide, pelo apoio incansável ao longo deste projeto.

À minha família, pelo amor incondicional, apoio constante e compreensão. Aos amigos, incluindo a incrível equipa da Shubox, pela amizade sincera, incentivo inestimável e por estarem sempre ao meu lado. A todas as pessoas que contribuíram na recolha de dados, o meu sincero e profundo agradecimento. Não menos importante gostaria de agradecer aos funcionários do bar do CP1 da Universidade do Minho, pela atenção e simpatia ao longo do meu percurso académico.

Este trabalho não teria sido possível sem o apoio de todos vocês. Obrigado do fundo do meu coração!

Resumo

Reconhecimento de atividades em *smartphones* usando sensores não intrusivos

O reconhecimento de atividades utilizando *smartphones* tem ganho uma atenção redobrada nos últimos anos devido à adoção generalizada destes dispositivos e consequentemente dos seus vários sensores. Estes sensores são capazes de fornecer dados bastante relevantes para este fim. Os sensores não intrusivos, em particular, oferecem a vantagem de recolher dados sem exigir ao utilizador a realização de quaisquer ações específicas ou o uso de dispositivos adicionais.

Primeiramente são discutidos os diferentes tipos de sensores habitualmente utilizados em *smartphones*, incluindo acelerómetros, giroscópios, entre outros, e o tipo de informação que nos podem guarnecer. Depois serão apresentados vários algoritmos e técnicas de *machine learning*, incluindo aprendizagem supervisionada, aprendizagem não supervisionada, e ainda aprendizagem por reforço. Dentro destes são também apresentados alguns dos algoritmos mais utilizados. Para finalizar esta primeira parte, serão ainda apresentadas alguns trabalhos realizados por outros investigadores na área.

O objetivo desta tese passou, então, pela criação de uma aplicação destinada ao reconhecimento de atividades recorrendo exclusivamente ao uso de sensores não intrusivos presentes em qualquer *smartphone*. Os dados colecionados por esses sensores forem submetidos a várias etapas de processamento e, após diversas iterações, obteve-se um conjunto de *features* altamente favoráveis ao treino dos modelos de *machine learning* criados. O melhor resultado foi obtido pelo modelo utilizando o algoritmo *XGBoost*, que alcançou uma impressionante taxa de *accuracy* de 0.979. Este resultado bastante sólido, permite verificar a alta eficácia do uso deste tipo de sensores para o reconhecimento de atividades.

Palavras-chave: Comportamento humano, Dispositivos móveis, *Machine learning*, Reconhecimento de atividades, Sensorização, *Smartphones*

Abstract

Activity recognition in smartphones using non-intrusive sensors

Activity recognition using smartphones has gained increased attention in recent years due to the widespread adoption of these devices and, consequently, their various sensors. These sensors are capable of providing very relevant data for this purpose. Non-intrusive sensors, in particular, offer the advantage of collecting data without requiring the user to perform any specific action or use any additional devices.

Firstly, we will discuss the different types of sensors commonly used in smartphones, including accelerometers, gyroscopes, among others, and the type of information they can provide us. Next, we will introduce several machine learning algorithms and techniques, including supervised learning, unsupervised learning, and reinforcement learning. Within these, some of the most commonly used algorithms will also be presented.To conclude this first part, some work carried out by other researchers in the field will also be presented.

The objective of this thesis was, therefore, the development of an application designed for activity recognition using exclusively non-intrusive sensors available in any smartphone. The data collected by these sensors underwent several processing stages, and after numerous iterations, a set of highly favorable features for training the machine learning models was obtained. The most prominent result was achieved by the model using the XGBoost algorithm, which achieved an impressive accuracy rate of 0.979. This quite robust result confirms the high effectiveness of using this type of sensors for activity recognition.

Keywords: Activity recognition, Human behavior, Machine learning, Mobile devices, Sensor data, Smartphones

Índice

Lista de Figuras

Lista de Tabelas

Glossário

accuracy Métrica de *performance* de *machine learning* cujo valor representa o número de previsões corretas a dividir pelo número de previsões.

f1-score Métrica de *performance* de *machine learning* cujo valor representa a média da *precision* com o *recall*.

KFold Fornece índices de treino/teste para dividir os dados em conjuntos de treino/teste. Divide os dados em k segmentos consecutivos. Cada segmento é depois usado uma vez como validação enquanto que os restantes segmentos são utilizados para o conjunto de treino.

precision Métrica de *performance* de *machine learning* cujo valor representa o número de previsões verdadeiras positivas dividido pela soma das verdadeiras positivas com falsas positivas.

recall Métrica de *performance* de *machine learning* cujo valor representa o número de previsões verdadeiras positivas dividido pela soma das verdadeiras positivas com falsas negativas.

Siglas

2D	Duas dimensões
ALS	Sensor de luminosidade ambiente
API	*Application programming interface*
CSV	*Comma-separated values*
EEG	Eletroencefalografia
HMM	*Hidden Markov model*
IoP	*Internet of people*
IoT	*Internet of things*
KNN	*K-nearest neighbors*
LCD	*Liquid-crystal display*
LMT	*Logistic model tree*
ML	*Machine learning*
MLP	*Multilayer perceptrons*
ORM	*Object Relational Mapping*
REST	*Representational State Transfer*
SARSA	*State-Action-Reward-State-Action*

SI Sistema Internacional

SMA *Signal-Magnitude Area*

SVM *Support Vector Machine*

WiFi *Wireless fidelity*

Capítulo 1

Introdução

1.1 Contexto & Motivação

Nos últimos tempos a utilização de *smartphones* tem vindo a aumentar de forma muito significativa [1]. Para além disso as pessoas estão cada vez mais sedentárias o que não favorece a sua saúde. Existem cada vez mais evidências que rastreadores de atividades físicas podem ajudar a saúde das pessoas [2], então porque não usar um dispositivo como o *smartphone*, que é já um item essencial no nosso dia a dia, para reconhecer as atividades que as pessoas realizam ao longo do dia?

Os *smartphones* possuem imensos sensores não intrusivos no seu interior [3], entre os quais o acelerómetro, o magnetómetro, giroscópio, sensores gravíticos, de luminosidade, e ainda estado da bateria. Através dos dados recolhidos por estes é então possível, usando modelos de *machine learning*, detetar diversas atividades que o utilizador destes dispositivos possa estar a realizar [4].

Desta forma qualquer pessoa poderia utilizar o seu próprio telemóvel para reconhecer as atividades por si praticadas não necessitando de utilizar outros dispositivos para o efeito.

1.2 Objetivos & Resultados esperados

O objetivo deste projeto foca-se essencialmente no estudo e identificação de atividades em *smartphones* recorrendo a sensores não-intrusivos. Assim o trabalho temo como base os seguintes objetivos:

- Desenvolver uma aplicação *mobile* capaz de colecionar dados;
- Explorar, estudar e tratar os dados colecionados;

- Conceber modelos de *machine learning* para previsão da atividade em execução por um utilizador do *smartphone*;
- Conceber um protótipo de software que utilize o melhor modelo.

A aquisição de dados relativos a diversas atividades, o seu estudo e tratamento, aliados ao desenvolvimento de modelos capazes de as caracterizar, representará uma grande parte do trabalho desenvolvido. O melhor modelo obtido será posteriormente incorporado num protótipo de *software* capaz de prever atividades em execução.

1.3 Estrutura do Documento

O seguinte documento encontra-se organizado em seis capítulos. O primeiro capítulo começa com uma introdução ao tema do estudo realizado. Em seguida é referido o contexto e motivação para o mesmo, assim como os objetivos e resultados que se esperam no final da dissertação. Para finalizar o capítulo existe ainda uma secção dedicada à apresentação da estrutura do presente documento.

No Capítulo 2 é apresentado o estado de arte desenvolvido com o propósito de descrever o que são sensores e também falar um pouco do que é *machine learning*, assim como mostrar algumas das técnicas mais conhecidas. No princípio é realizada uma apresentação do que é a sensorização, assim como apresentados diversos sensores que podem ser encontrados em *smartphones* comuns e que se possam revelam interessantes para o estudo. Em seguida serão apresentados diversos tipos de paradigmas de *machine learning* assim como alguns dos algoritmos mais utilizados para cada um deles.

Em seguida, no terceiro capítulo será apresentado a *Activity Tracker* uma aplicação desenvolvida com a finalidade de colecionar dados. Primeiramente é apresentada a forma como foi realizado o seu desenvolvimento *Android*. Em seguida é apresentada a aplicação criada com a finalidade de ser um servidor disponível 24 horas por dia, ao qual a aplicação *Android* envia os seus dados, permitindo assim o seu armazenamento. Para finalizar o capítulo, existe ainda uma secção que apresenta os dados obtidos da interação destas duas anteriores.

No quarto capítulo é primeiramente abordada a metodologia usada para a aplicação de previsão e para a arquitetura final. Em seguida, é descrito o processo de preparação dos dados, que engloba técnicas como balanceamento dos dados, identificação e eliminação de valores atípicos, bem como a remoção de *features* que poderiam introduzir tendências indesejadas nos resultados. Na sequência, são apresentadas as experiências conduzidas para cada um dos modelos desenvolvidos, incluindo informações sobre as configurações experimentais e os parâmetros selecionados. Por fim, o capítulo é encerrado com um sumário que destaca o conjunto final das *features* utilizadas nos modelos.

No seguimento, o Capítulo 5 apresenta uma análise dos resultados obtidos. Inicialmente, é discutida a aplicação de dados desenvolvida, avaliando o seu desempenho no contexto do estudo. Em seguida,

é realizada uma análise individual minuciosa para cada modelo, examinando o seu desempenho relativamente a cada uma das suas classes, além da análise das *features* com maior importância em cada modelo. Para encerrar o capítulo é apresentada uma breve comparação entre os valores de *accuracy* finais dos modelos desenvolvidos.

Para finalizar este documento, o Capítulo 6 não apenas contém uma conclusão, mas também apresenta várias ideias de trabalho futuro e algumas reflexões finais sobre o trabalho desenvolvido.

Capítulo 2

Estado de Arte

Este capítulo introduz fundamentos teóricos assim como um plano mais técnico, necessários para melhor compreender o tema desta dissertação. Deste modo serão referidos essencialmente dois grandes temas fundamentais para a realização do projeto, a sensorização e *machine learning*.

2.1 Sensorização

A sensorização é um conceito bastante amplo, de modo que para uma melhor compreensão do mesmo, serão primeiramente apresentados alguns conceitos chave que se encontram inerentemente interligados a este conceito mais geral.

- **Inteligência ambiente** - refere-se a um meio ambiente que tem a capacidade de sentir e responder à presença de pessoas, isto de forma não intrusiva. Como cada vez mais os dispositivos eletrónicos comunicam entre si, são mais pequenos e se integram melhor no meio ambiente, percebe-se facilmente que a tecnologia acaba por desaparecer nas nossas redondezas [5].

- ***Internet of Things* (IoT)** - é o nome que se dá a uma rede de objetos inteligentes que tem a capacidade de se organizarem, partilhar informação, dados, recursos e ainda reagir face a mudanças acontecidas à sua volta [5].

- ***Internet of People* (IoP)** - representa uma rede global dinâmica constituída por pessoas e dispositivos que se compreendem e comunicam entre si. É onde tudo e todos conseguem sentir a presença dos outros e agir na presença de informação e conhecimento com o objetivo de melhorar a qualidade de vida das pessoas [5].

- ***Smart Cities*** - representa um lugar onde os serviços e redes tradicionais são melhorados em termos de eficiência através do uso de soluções digitais, favorecendo não só os habitantes como empresas. Estas melhorias podem dar-se a nível de redes de transporte inteligentes, melhorias na distribuição de água, maior eficiência na iluminação e aquecimentos de edifícios e ainda uma uma administração de uma cidade mais rápida e interativa [6].

- ***Pirâmide dos dados*** - simboliza a hierarquia do que são dados e como se organizam. Esta hierarquia possui quatro conceitos base, os dados, a informação, o conhecimento e a sabedoria. Uma pirâmide representativa desta hierarquia pode ser visualizada na figura 1.

  - Os **dados** encontrando-se na base da pirâmide, referem-se a uma coleção de factos de forma não organizada nem processada. São exemplos de dados: vermelho, 2312, 23.67, ativo, entre outros.
  - A **informação**, situada no nível seguinte acrescenta significado aos dados, facilitando a sua medição, visualização e análise num dado contexto. É exemplo de informação: O semáforo da rua do estádio passou a vermelho.
  - O **conhecimento** segue-se à informação, fornecendo-lhe contexto. Pegando no exemplo de informação anterior, pode passar a conhecimento da seguinte forma: O semáforo para o qual me estou a dirigir passou a vermelho.
  - A **sabedoria** localizada no topo da pirâmide, serve-se do conhecimento e aplica-o numa dada situação. Assim um exemplo seria: Já que o semáforo para o qual me estou a dirigir está vermelho, é melhor parar!

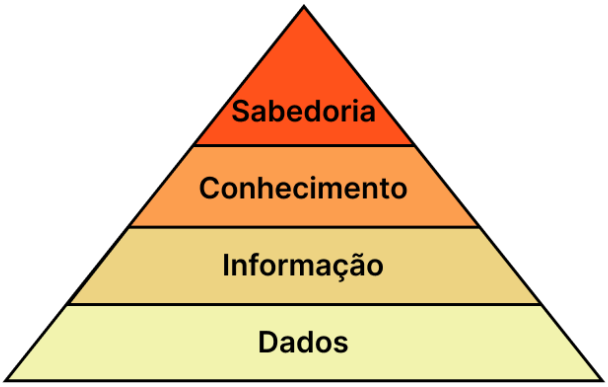


Figura 1: Pirâmide dos dados

A sensorização acaba por ser a aglomeração de sensores. Já um sensor é algo capaz de percecionar um fenómeno que esteja a observar e posteriormente transmitir o seu estado [5]. Os sensores como iremos ver mais à frente, estão subdivididos em dois grupos, os sensores físicos e os sensores virtuais.

2.1.1 Sensores Físicos

Um sensor físico, tal como o próprio nome indica, é um sensor que existe no mundo físico. Este por sua vez é um dispositivo utilizado para medir quantidades físicas que são convertidas em sinais (normalmente sinais elétricos) que podem ser posteriormente interpretados por um observador ou por um outro dispositivo [7].

Existe uma diversidade enorme de sensores deste tipo, cada um, claro está, com o seu devido propósito. Este tipo de sensores é muito utilizado tanto na IoT como em robôs que funcionam através de placas baseadas em *Arduino*. Existem ainda outros dispositivos deste tipo, designados de *beancons*, que tendo como base o *bluetooth* permitem realizar *geofencing*, isto é detetar dispositivos que entram numa determinada área.

Todos nós sem darmos conta, carregamos todos os dias imenso sensores no bolso. Um *smartphone* atual possui uma variedade enorme de sensores físicos, cada um com um papel fundamental para realizar funções que damos como garantidas em telemóveis atuais. Desta forma, destaco alguns destes sensores juntamente com uma das suas funções:

- **Câmara fotográfica** - tira fotografias;
- **Sensor de luminosidade** - deteta intensidade da luz;
- **Termómetro** - mede a temperatura interna do dispositivo;
- **Sensor de impressão digital** - permite a autenticação no telemóvel;
- **Microfone** - capta o áudio do utilizador;
- **Acelerómetro** - permite o telemóvel mudar a orientação;
- **Magnetómetro** - mede a força do campo magnético para determinar a direção do movimento;
- **Sensor de proximidade** - impede toques acidentais durante uma chamada.

Hoje em dia existem sensores físicos em todo o lado, mesmo sem nos apercebermos. Estes sensores permitem tornar o nosso mundo cada vez mais inteligente, já que os dados que fornecem possibilitam a tomada de decisões por parte de sistemas, tendo em conta o ambiente ao seu redor.

2.1.2 Sensores Virtuais

Como vimos anteriormente, sensores físicos são aqueles que reagem a estímulos físicos do ambiente e com isso enviam sinais que podem ser captados em formato digital. Ao contrário destes, um sensor virtual é puramente baseado em *software*. Estes sensores produzem sinais de forma autónoma através

da agregação e combinação de outros sinais que estes recebam a partir de outros sensores, quer estes sejam físicos ou virtuais [8]. Um sensor virtual não pode existir caso não exista qualquer sensor físico previamente a captar dados. Nas figuras 2, 3 e 4 podemos ver algumas redes que representam as fontes dos sinais que chegam aos sensores virtuais.

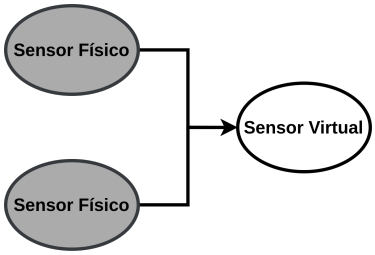


Figura 2: Rede 1 - exemplo de encaminhamento de sinais entre sensores (adaptado de [8])

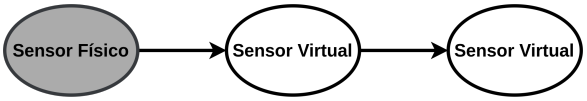


Figura 3: Rede 2 - exemplo de encaminhamento de sinais entre sensores (adaptado de [8])

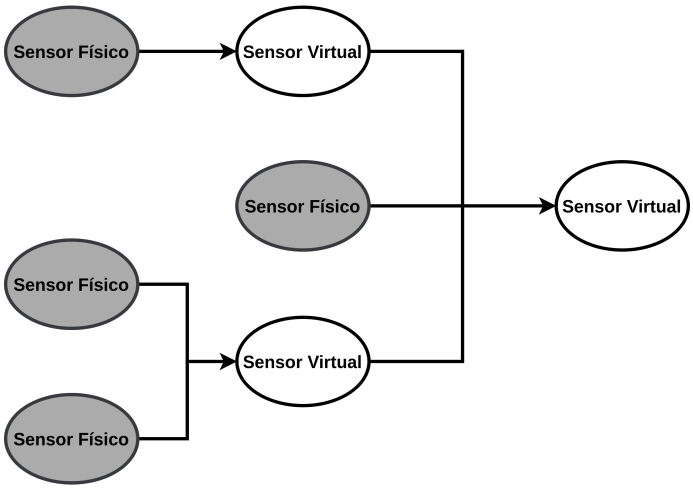


Figura 4: Rede 3 - exemplo de encaminhamento de sinais entre sensores (adaptado de [8])

Assim tipos de *software* que forneçam resultados através do processamento de dados recebidos por um conjunto de sensores, podem ser considerados sensores virtuais. Deste modo podemos afirmar, por exemplo, que são sensores deste tipo qualquer *software* que realize medições relativas à utilização de um rato de computador, como total distância percorrida, número de cliques, velocidade média, entre outros. Podem ainda ser denominados de sensores virtuais, serviços chamados através de APIs que forneçam dados sobre qualquer tópico, como por exemplo meteorologia, poluição do ar ou até informações de trânsito.

2.1.3 Sensores

2.1.3.1 Acelerómetro

Um acelerómetro fornece um conjunto de três valores na forma (x,y,z), que representam as forças que estão a atuar no dispositivo [9], quer estas sejam estáticas, como é o caso da força gravítica, quer sejam dinâmicas como é o caso das forças sentidas aquando da realização de um movimento [10]. Estes dados são representados relativamente a cada um dos seus eixos cartesianos, x,y e z [9, 11], como se pode observar na figura 5. A unidade das medições realizadas por este sensor encontram-se normalmente de acordo com a unidade SI da aceleração, m/s^2. Quando um acelerómetro se encontra em repouso os seus valores irão tender na direção de igualar a força da gravidade que neles se encontra a ser exercida [4].

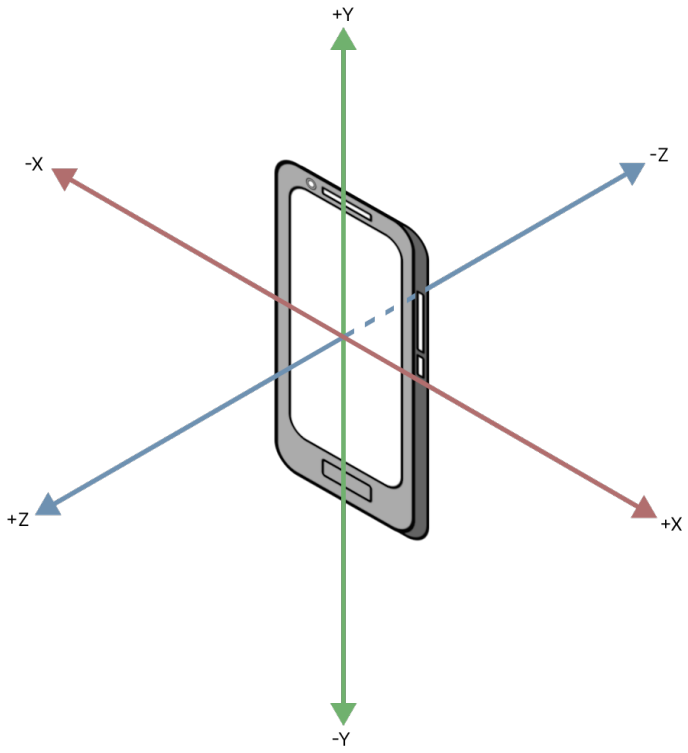


Figura 5: Eixos de um acelerómetro (adaptado de [4])

Os acelerómetros podem ser usados para diversos fins. Em computadores portáteis mais antigos, utilizando ainda discos rígidos, um sensor deste tipo era utilizado para detetar quedas. Este deteção permitia que o disco fosse imediatamente desligado e assim permitiria reduzir os danos a este causados pela queda. Também em carros podem ser encontrados sensores deste tipo. Aquando de um acidente um acelerómetro permite a sua deteção e consequente ativação dos *airbags*. No que toca a *smartphones* este é o responsável pela rotação automática do ecrã quando o utilizador inclina o telemóvel [12].

Embora um sensor deste tipo possa parecer simples, estes possuem diversas partes diferentes que

necessitam de interagir entre si para realizarem uma boa leitura. Os acelerómetros mais comuns funcionam através do efeito piezoelétrico. Este efeito pode ser resumido pela criação de voltagem por parte de cristais microscópios quando deformados pela ação das forças que neles atuam. A voltagem gerada pode depois então ser interpretada para determinar a velocidade e orientação [12].

2.1.3.2 Giroscópio

Também o giroscópio fornece uma leitura a três dimensões, isto é, um conjunto de valores (x,y,z) que representam uma perspetiva da orientação do dispositivo. Este sensor calcula a rotação do dispositivo em rad/s face a cada um dos eixos. Os valores fornecidos por este variam entre positivos, quando o dispositivo está a virar no sentido dos ponteiros do relógio, ou negativos, no caso em que esteja virado no sentido contrário aos [4], como demonstrado na figura 6.

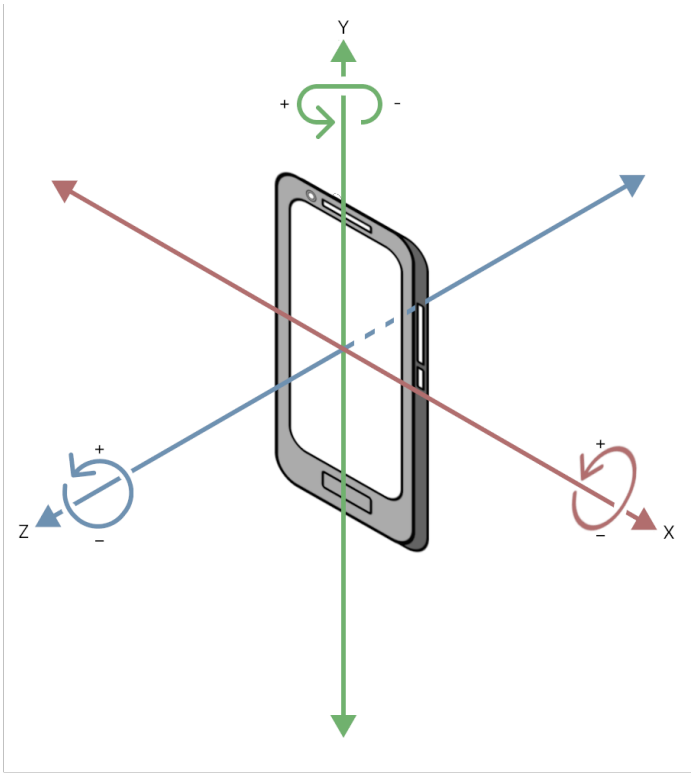


Figura 6: Eixos de um giroscópio (adaptado de [4])

Este tipo de sensores pode ser usado para múltiplos propósitos. Num primeiro estudo [13], os investigadores usaram este tipo de sensor juntamente com EEG e processamento de imagem para detetar a fadiga de uma pessoa enquanto conduz um automóvel. Num segundo estudo, este foi utilizado para detetar a subida de escadas durante atividade física [14]. Giroscópios podem ainda ser utilizados para estabilizar imagens de câmaras, detetar rolamento de automóveis e ainda podem ser usados em sistemas de navegação.

O funcionamento de muitos giroscópios baseia-se na deteção da aceleração induzida pela força de Coriolis numa massa que vibra numa direção ortogonal ao eixo sobre o qual a rotação é aplicada. O cálculo da rotação é então estimado medindo o deslocamento dessa massa face ao movimento rotacional e ao eixo em que a rotação é suposto ser sentida [13].

2.1.3.3 Sensor Gravitacional

Um sensor gravitacional também fornece os seus dados sob a forma (x, y, z), medindo os vetores das componentes da gravidade relativamente ao um dispositivo [15]. Tal como os acelerómetros, a unidade medida por estes dispositivos é medida em m/s^2 [16]. Na figura 7 observa-se uma seta amarela que representa o vetor da gravidade, sendo que o valor apresentado por estes dispositivos será uma decomposição deste vetor ao longo dos eixos cartesianos relativos do dispositivo.

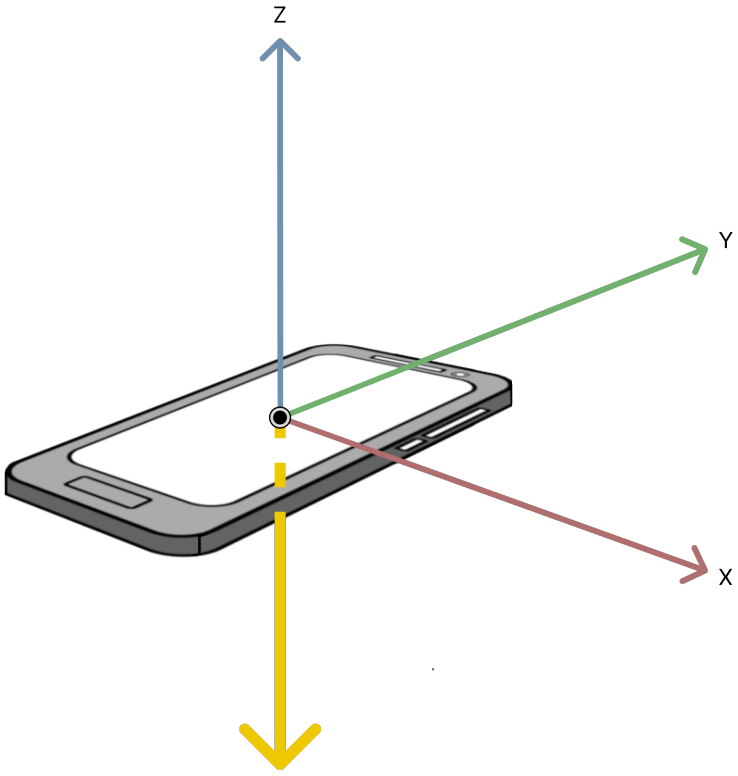


Figura 7: Eixos de um sensor de gravidade (adaptado de [17])

Sensores gravitacionais podem ter diversos usos, quer para detetar ações do dia a dia quer em estudo dos solos. Estes sensores conseguem captar por exemplo o movimento de uma perna ao subir escadas, já que este movimento altera a posição do dispositivo face ao vetor da gravidade [15]. Podem ainda servir para encontrar reservas minerais em profundidade, uma vez que a diferença de densidade entre os materiais provoca uma alteração dos valores registados, de modo que é possível saber-se o que se encontra no subsolo [18].

Em *smartphones Android* é raro haver sensores físicos deste tipo, normalmente este tipo de sensores encontram-se na forma de um sensor virtual, uma vez que é costume ser um *software* que tira proveito de um acelerómetro físico para realizar estas medições [16].

2.1.3.4 Sensor de Orientação

Por sua vez o sensor de orientação, recorre também a um sistema de três eixos (x, y, z), para representar a orientação do dispositivo através do ângulo em graus, que este faz com cada um dos eixos. O ângulo em torno do eixo X representa a azimute, que apresenta os valores de 0°, 90°, 180°, 270° para representar quando se aponta na direção norte, este, sul e oeste respetivamente. Por sua vez o ângulo em torno do eixo do Y é representativo da inclinação do dispositivo, apresentando os valores 0°, -90°, 90° e 180° ou -180° quando se encontra respetivamente pousado virado para cima, quando está em pé, quando está de cabeça para baixo e quando está pousado virado para baixo. Já o eixo Z representa o ângulo de rotação, representando a inclinação lateral em valores compreendidos entre -90° e 90°, sendo que quando está totalmente rodado para a esquerda o valor é -90°, quando está rodado para a direita é 90° e o valor apresentado por estar orientado para cima é 0 [19]. As rotações podem ser observadas de forma ilustrativa na figura 8.

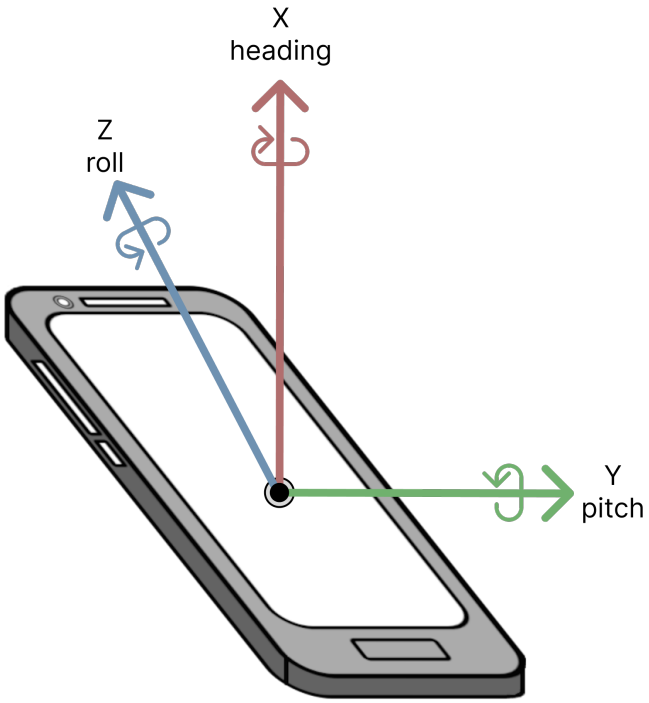


Figura 8: Eixos de um sensor de orientação (adaptado de [19])

Sensores deste tipo podem ser usados para diversas finalidades. No artigo [20] um sensor deste tipo juntamente com um sensor de posicionamento, foram utilizados para estimar volumes a partir de imagens de ultrassom 2D multiplanares. Relacionados com deteção de movimentos em seres humanos,

no artigo [21] é referido como se pode usar sensores de orientação para monitorizar o comportamento humano enquanto se conduz um veículo. Um outro exemplo, desta vez mais simples, os dados relativos aos valores do eixo X deste tipo de sensor, pode ser usado na construção de uma bússola baseada em *software*.

Em *smartphones*, mais concretamente em *smartphones Android*, os dados fornecidos por este sensor resultam do processamento de dados recebidos pelo acelerómetro e pelo sensor de campo geomágnético presentes no mesmo dispositivo [16].

2.1.3.5 Sensor de Luminosidade

Os sensores de luminosidade, também chamados de ALS, servem essencialmente para medir a intensidade da luz que incide sobre eles. A resposta espectral dos ALS, semelhante à do olho humano, situa-se entre os 350 nm e os 1050 nm [22]. Deste modo estes sensores apresentam apenas um valor para representar a sua medição, sendo a unidade utilizada chamada Lux [23]. Na tabela 1 pode-se perceber como funciona esta unidade face a diferentes luminosidades encontradas no dia a dia.

Categoria	luminosidade (Lux)
1 vela a 1 metro de distância	1
Luz da estrada	20
Iluminação de mesa de escritório	750
Dia nublado	3000
Dia ensolarado nublado	20 000
Luz direta do sol	100 000

Tabela 1: Escala de luminosidade (adaptado de [23])

Em todas as situações que seria útil alterar a luminosidade de algo baseando-se na luz ambiente, este tipo de sensores pode ser utilizado. Permite poupar bateria em diversos dispositivos com ecrãs LCD, visto que é possível com estes sensores um ajuste automático da sua luz de fundo. Esta regulação da luz de fundo permite também um escurecimento automático do ecrã para manter a aparência sob diferentes ambientes de luminosidade. Esta mesma tecnologia de escurecimento automático pode também ser usada em automóveis, candeeiros de escritório, luzes exteriores e sinais de trânsito [23].

Normalmente os sensores de luminosidade ambiente que também são conhecidos como fotodiodos, são aparelhos que conseguem converter a luz incidente em tensão ou corrente elétrica. Quando um fotão com energia suficiente atinge o diodo, este fotão irá dividir-se em partículas que em seguida são aceleradas em direções contrárias. Este movimento das partículas é o responsável pela criação da corrente elétrica gerada no fotodiodo [24]. Em *smartphones* este tipo de sensor costuma ser colocado na sua parte frontal (figura 9) de modo a fornecer dados mais relevantes relativamente à luz incidente no ecrã.

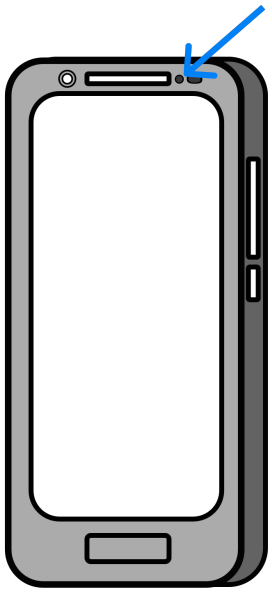


Figura 9: Localização habitual do sensor de luminosidade num telemóvel

2.1.3.6 Sensor de Proximidade

Um sensor de proximidade, como o próprio nome indica, fornece informação relativa à proximidade de objetos face a este. No caso de *smartphones*, este tipo de sensor, normalmente situado na sua parte frontal, como se pode observar na figura 10, fornece a distância em cm de um objeto relativamente ao ecrã [16].

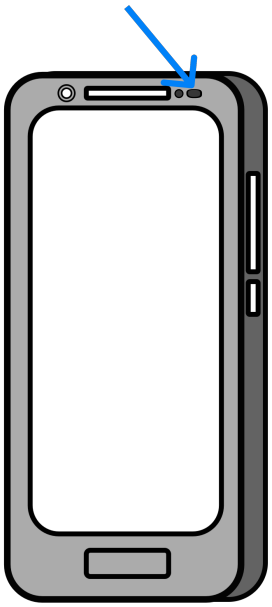


Figura 10: Localização habitual do sensor de proximidade num telemóvel

Em telemóveis comuns, é usado frequentemente para detetar a distância face à orelha quando um

utilizador atende ou desliga uma chamada. No caso do atendimento de uma chamada, o sensor deteta a proximidade da orelha e desliga o ecrã para evitar toques acidentais. Quando o sensor deteta o afastamento face à orelha, volta a ligá-lo [25]. Este mesmo comportamento pode ser obtido quando se coloca o dispositivo no bolso. Estes sensores podem também ser encontrados nos veículos que conduzimos diariamente. Os mecanismos de apoio ao estacionamento utilizam estes sensores para poder avisar o condutor da excessiva proximidade de algum objeto aquando da execução das manobras [26]. Existem ainda outros tipos de sensores de proximidade que conseguem detetar a proximidade de metais, sendo este tipo usado para por exemplo detetar minas explosivas ou outro tipo de metais enterrados [27].

Existem diversos tipos de sensores de proximidade, no entanto, em telemóveis costumam ser utilizados sensores de proximidade fotoelétricos [25]. Para detetar objetos na sua frente, este tipo de sensores age de forma parecida a um sonar, só que em vez de enviar ondas sonares, estes emitem uma luz que é novamente captada após refletir na superfície de algum objeto [28].

2.1.3.7 Sensor de *Bluetooth*

O *Bluetooth* é uma tecnologia usada para transferências de informação a curtas distâncias. Esta tecnologia permite a partir de ondas na frequência das ondas rádio transmitir documentos ou manter a conexão entre dispositivos que possuam ligação deste tipo [29]. Os sensores de *Bluetooth* em *smartphones* conseguem captar bastante informação à cerca da utilização desta tecnologia. Alguns exemplos encontram-se em seguida [30]:

- Detetar o número de dispositivos *bluetooth* detetados nas redondezas;
- Visualizar a proporção entre dispositivos novos nas redondezas;
- Visualizar a proporção entre dispositivos antigos nas redondezas;
- Calcular a média da taxa de mudança de novos dispositivos nas redondezas;
- Calcular o desvio padrão da taxa de mudança de novos dispositivos nas redondezas.

Este tipo de dados permite fornecer diversas informações sobre o dono do telemóvel, que no caso do estudo [30], estes dados foram usados com o objetivo de utilizando algoritmos de *machine learning*, reconhecer vários tipos de ambientes de contexto social.

2.1.3.8 Sensor de Conectividade

Os sensores de conectividade, na sua forma mais simples, servem para detetar o tipo de conexão à *internet* que um dispositivo apresenta no momento da recolha dessa informação. Assim um *smartphone* pode estar num dos seguintes estados:

- Ligado através de WiFi;
- Ligado a partir de dados móveis;
- Desconectado da *internet*.

Estando um utilizador em movimento e com o WiFi ligado, através do número de WiFi *scans* únicos e da localização de redes WiFi, é possível detetar padrões de mobilidade [31].

2.2 *Machine Learning*

Machine learning de uma forma resumida, é um sistema que aprende a partir de dados que lhe são fornecidos, ao invés do uso de programação explícita. Este ramo da inteligência artificial possui uma variedade vasta de algoritmos que, de forma interativa, conseguem aprender com dados para os descrever, melhorar e ainda prever os seus resultados. Quando um algoritmo de *machine learning* é treinado com dados, este gera um *output* que é denominado de modelo. Depois deste modelo estar treinado, é possível fornecer-lhe dados novos aos quais este irá apresentar um resultado. Em caso de modelos de previsão, após receberem dados, estes irão gerar um resultado de acordo com os dados que foram usados para os treinar [32].

Todos nós já alguma vez interagimos com aplicações de *machine learning* no nosso dia a dia. Quando vamos a um *site* de comércio ou usamos aplicações de supermercados, todas as sugestões que nos são apresentadas não são inocentes. Todas estas recomendações são-nos geradas por modelos de *machine learning* que aprenderam com as nossas pesquisas ou comprar anteriores, para nos darem recomendações mais relevantes para no final conseguirem fazer com que o consumidor acabe por comprar mais alguma coisa, mesmo que este não precise dela.

2.2.1 Aprendizagem com Supervisão

A aprendizagem com supervisão começa normalmente com um conjunto de dados pré estabelecidos que apresentam alguma compreensão relativamente à sua classificação. Este paradigma destina-se então a encontrar padrões nos dados que podem depois ser utilizados em processos analíticos. Todos os dados aqui presentes, estão rotulados de acordo com o seu significado, por exemplo em dados de imagens de animais, existira um valor representativo do nome do animal, permitindo assim a uma pessoa que treina os dados identificar facilmente se o modelo está a seguir no melhor sentido. Este rótulo pode ser um dado de natureza discreta tornando-se um problema de classificação, ou de natureza contínua, também conhecido como um problema de regressão [32]. São exemplos de problemas de classificação a classificação de imagens, deteção de fraude, e diagnósticos. Já a previsão meteorológica, previsão de

mercados, crescimento populacional ou ainda esperança média de vida, são exemplos de problemas de regressão [33].

Os algoritmos deste paradigma usam exemplos de dados pré-processados para treinar e posteriormente eles são avaliados com dados de teste. Algo que é preciso ter em atenção é não utilizar os mesmos dados para treinar e para testar, uma vez que isto pode resultar em *overfitting*, isto é, o modelo decorar os casos que treinou, e depois não ser capaz de generalizar para casos novos que lhe apareçam. Usar dados não já utilizados no teste não servem apenas para isto, mas também para avaliar a precisão do modelo em prever resultados. Para além dos exemplos já fornecidos, modelos com aprendizagem supervisionada podem ainda ser usados em soluções de recomendação, reconhecimento de voz ou até análise de riscos [32]. Alguns exemplos de algoritmos utilizados neste tipo de aprendizagem podem ser vistos com mais detalhe em seguida.

2.2.1.1 Árvores de decisão

Como o próprio nome indica é baseado numa árvore, isto é, num grafo hierarquizado. Cada ramo representa uma seleção de entre diversas alternativas, e cada folha representa uma classe. Na figura 11 apresenta-se um exemplo de uma árvore de decisão relativa a um jantar com amigos. Primeiro dependendo das horas, caso sejam 19 horas ou mais cedo, não se irá jantar, depois caso seja para jantar em casa também não se realizará o jantar. Este só se realizará caso seja depois das 19 horas e num restaurante.

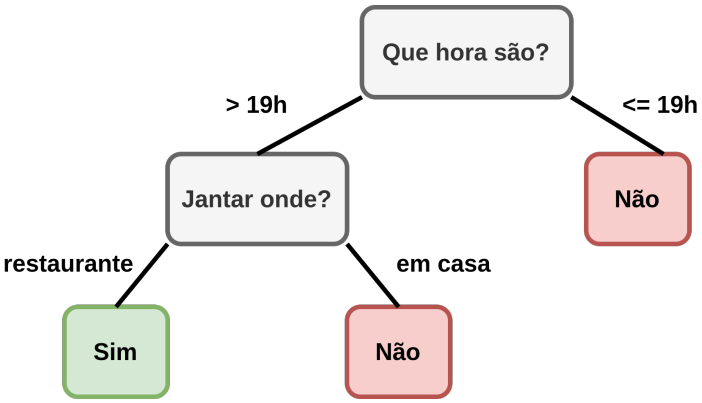


Figura 11: Exemplo de árvore de decisão (adaptado de [34])

Existem árvores de decisão para problemas de classificação, em que os resultados são discretos, como é o caso do exemplo anterior, classificar a fruta representada numa imagem ou ainda a sobrevivência ou não num acidente de aviação. Também se pode aplicar este algoritmo a problemas de regressão para problemas como a previsão do tráfego rodoviário ou a previsão de preços de ações [34].

2.2.1.2 Regressão linear

Este algoritmo tenta prever um resultado Y baseado numa variável x. Tenta simplificar o problema a uma linha reta, usando-a para prever valores não observados. Esta reta segue a fórmula da reta $Y = m.x + b$ sendo m o valor do declive da reta e b o valor marcado na interceção com o eixo y. Para calcular esta reta é normalmente utilizada a minimização dos erros ao quadrado entre cada ponto e a reta [35]. Na figura 12 pode ser observado um exemplo de um modelo de regressão linear. Este tipo de algoritmo é apenas usado em problemas de regressão devido à sua natureza.

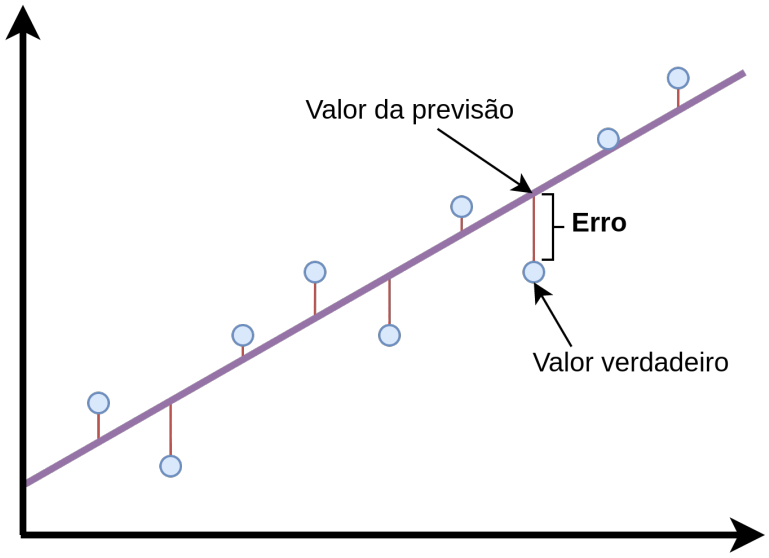


Figura 12: Exemplo de regressão linear

2.2.1.3 Regressão logística

A regressão logística em contraste à regressão linear é utilizada para problemas de classificação binária, isto é realizar a previsão de duas categorias discretas. Os modelos normais de regressão linear não se iriam adaptar bem na regressão logística, de modo que a regressão linear é transformada numa curva de regressão logística. Para isso é utilizada a função *sigmoid* $S(x) = \frac{1}{1+e^{-x}}$ que pega num valor e transforma-o num valor entre 0 e 1, representando assim a probabilidade entre 0 e 1 de pertencer a uma classe. Definindo o 0.5 como limite entre as classes, valores inferiores a 0.5 serão classificados com a classe 0 e valores superiores com a classe 1 [35]. Pode-se observar um exemplo de um modelo deste tipo na figura 13.

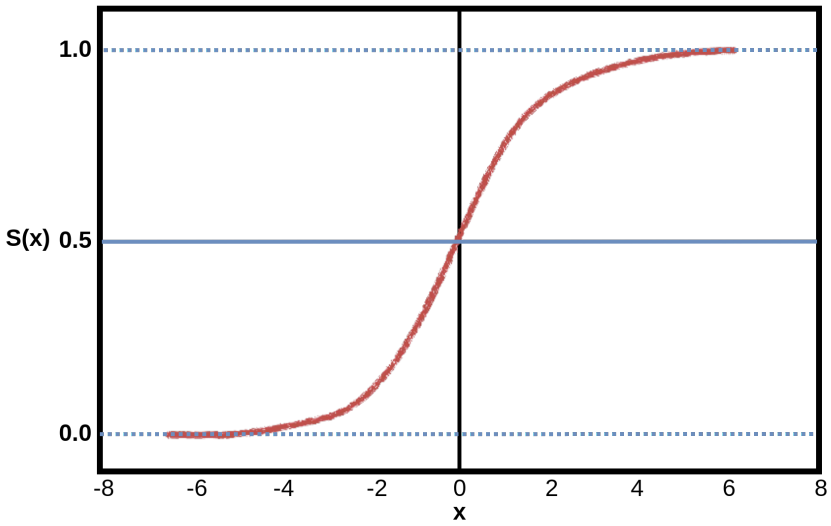


Figura 13: Exemplo de regressão logística (adaptado de [35])

2.2.1.4 *Support Vector Machines* (SVM)

O SVM é um algoritmo que pode ser usado quer para problema de classificação quer para problemas de regressão. A essência deste algoritmo é desenhar cada ponto num espaço de dimensão n realizando a classificação encontrando o hiperplano que melhor diferencia as classes. Os hiperplanos são barreiras que ajudam a classificar os dados. Caso o número de *features* for dois, este hiperplano seria uma linha, caso fosse três o hiperplano seria um plano e por aí em diante. Um ponto pertencerá à classe cujo espaço inclua esse mesmo ponto. Na figura 14 encontra-se o exemplo de um modelo SVM com 2 *features* [36].

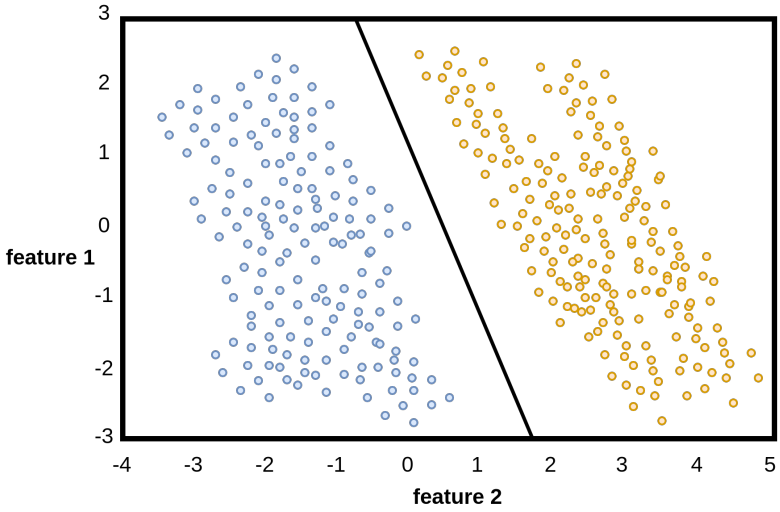


Figura 14: Exemplo de *Support Vector Machine* com 2 *features* (adaptado de [36])

2.2.1.5 Rede neuronal

Redes neuronais são modelos de *machine learning* que seguem uma analogia face ao cérebro humano. Uma rede neuronal é um processo paralelo composto por unidades de processamento simples. O conhecimento é guardado nas conexões entre estas unidades e é adquirido a partir de dados, através de processos de aprendizagem que ajustam o peso entre as suas conexões.

Estas unidades de processamento simples, também conhecidas como neurónios, recebem um conjunto de *inputs* e têm um peso associado a cada conexão que estabelecem. Cada neurónio calcula a sua ativação baseado nos dados de *input* e nos pesos da conexão. Este sinal calculado é posteriormente passado para o *output* do neurónio depois de ser filtrado por uma função de ativação.

Existem numerosos tipos de arquiteturas de redes neuronais, podendo ser tanto utilizados para aprendizagem com supervisão quer em aprendizagem sem supervisão (que iremos ver mais à frente). Redes neuronais com diversas camadas intermédias como o exemplo da figura 15, podem-se chamar de redes neuronais profundas, enquanto que aquelas com apenas uma rede intermédia podem ser chamadas de *Multilayer perceptron* ou MLP.

Em modelos de aprendizagem supervisionada os dados são constituídos por exemplos de treino que possuem o seu *input* e o seu *output* desejado. Já o objetivo do modelo passará por corrigir os valores dos pesos das conexões do neurónios, de modo a minimizar a função de custo. Um dos algoritmos mais utilizados denomina-se *backpropagation*. Este algoritmo possui duas fazes, a primeira *forward propagation* que calcula tanto o valor de *output* para um vetor de *inputs*, tanto o valor do erro cometido. A segunda fase é o que dá o nome a este algoritmo (*backpropagation*) que irá propagar para trás o erro cometido, ajustando os pesos das conexões de modo a diminuir o seu valor. É baseado no cálculo do gradiente através do uso da regra da cadeia de funções compostas [37].

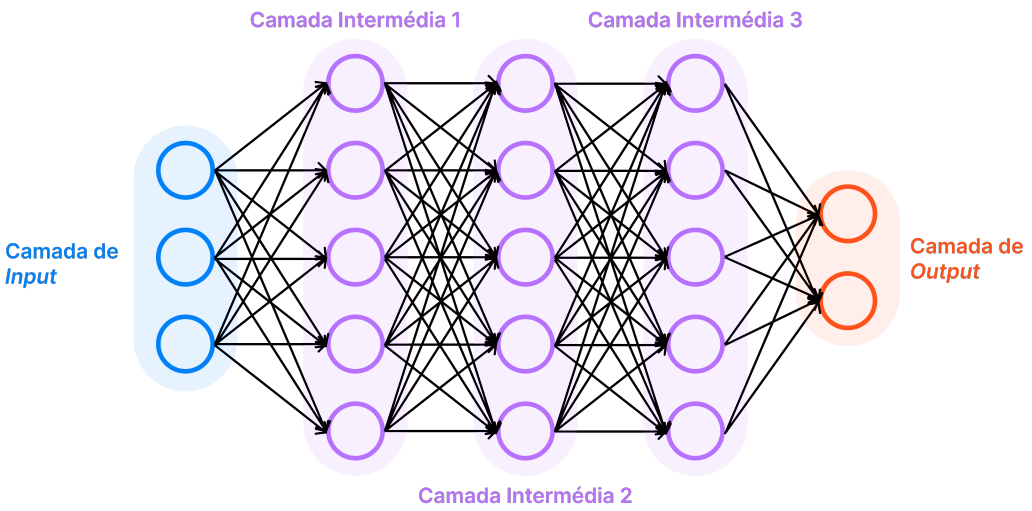


Figura 15: Representação de rede neuronal com 3 camadas intermédias

2.2.2 Aprendizagem sem Supervisão

A aprendizagem sem supervisão é mais utilizada em situações em que existem enormes quantidades de dados não nomeados. Os algoritmos deste paradigma tentam compreender o significado dos dados a partir da classificação dos dados a partir de padrões ou em *clusters*. Deste modo a aprendizagem sem supervisão, como o próprio nome indica, conduz um processo interativo sem a intervenção de um ser humano [32]. Os problemas da aprendizagem sem supervisão podem-se dividir em duas categorias, a segmentação que é usada quando se pretende distribuir os dados por grupos coerentes, e a associação cujo objetivo é associar dados com comportamento semelhante. São exemplos de problemas de segmentação sistemas de recomendação, *marketing* e segmentação de clientes. No que toca a problemas de associação são exemplo a visualização de *big data*, compreensão de significados, seleção de atributos ou ainda descoberta de estruturas [33].

Os algoritmos de aprendizagem sem supervisão segmentam os dados em grupos de exemplos (*clusters*) ou em grupos de características. De modo a que os dados sejam rotulados, e assim poderem-se tornar em dados supervisionados, os dados não rotulados criam parâmetros e a sua própria classificação. Na presença de enormes quantidades de dados isto é possível, permitindo ter o primeiro passo para dados usados em aprendizagem supervisionada, dados estes que não seriam possíveis de ser identificados pelo desenvolvedor do modelo, visto a impossibilidade de este conseguir saber o contexto dos dados analisados [32].

Semelhante aos algoritmos da aprendizagem supervisionada, os algoritmos não supervisionados também procuram padrões nos dados, a grande diferença e que nesta segunda, os dados não são compreendidos a priori. No exemplo da saúde, colecionar quantidades enormes de dados sobre uma doença específica pode ajudar a relacionar padrões de sintomas e associá-los com os resultados dos seus pacientes. Iria demorar uma eternidade para conseguir rotular todos estes dados, daí a importância da utilização da aprendizagem não supervisionada [32].

2.2.2.1 *K-Means*

O *k-means* é uma das heurística utilizada em algoritmos de particionamento. Estes algoritmos têm como objetivo atingir um ótimo global através da enumeração exaustiva de todas as partições. Dado um conjunto de dados D e um número de segmentos k, o que os algoritmos fazem é partir D em k segmentos. A utilização do *k-means* baseia-se nos seguintes 4 passos [38]:

1. Dividir os objetos em k subconjuntos contendo pelo menos um elemento;
2. Calcular o *centroid*, o centro de cada um dos subconjuntos;
3. Atribuir cada elemento ao *centroid* mais próximo de si;
4. Voltar para o segundo passo até não houver mais atribuições possíveis.

2.2.2.2 *K-Medoids*

A heurística *k-medoids* é um pouco semelhante à anterior, só que em vez de *centroids* temos *medoids*. A grande diferença é que os *medoids* em vez de serem pontos aleatórios, são objetos representativos do conjunto de dados. Para aplicar esta heurística podem-se seguir os seguintes passos [38]:

1. Selecionar *k medoids* representativos de objetos dos dados;
2. Medir a distância dos dados a cada *medoid*;
3. Atribuir cada elemento ao *medoid* mais próximo de si;
4. Selecionar como novo *medoid* o elemento de dados mais central de cada sub-conjunto;
5. Voltar para o segundo passo até não ser possível encontrar novos *medoids*.

2.2.2.3 *Autoencoder*

Um *Autoencoder* é um tipo de rede neuronal utilizado para aprender uma representação compacta dos dados de *input*, chamada de *encoding*. O objetivo de um *Autoencoder* é reduzir a dimensionalidade dos dados de *input* tentando sempre preservar a informação original ao máximo. Os *autoencoders* são tipicamente utilizados como parte de um processo de aprendizagem sem supervisão, onde os dados não se encontram rotulados, com o objetivo de aprender a estrutura subjacente a esses mesmos dados.

Os *autoencoders* podem ser usados para resolver diversos problemas, entre os quais reconhecimento facial, deteção de anomalias, deteção de características e ainda aquisição do significado das palavras. Estes modelos são por sua vez modelos generativos que podem gerar dados novos de forma aleatória que sejam semelhantes aos dados de *input* presentes nos dados de treino.

2.2.3 Aprendizagem por Reforço

A aprendizagem por reforço já apresenta maiores diferenças face às duas anteriores. Os algoritmos deste tipo de aprendizagem recebem *feedback* através da análise dos dados, para de forma guiada chegar ao melhor resultado. Aqui os dados não são treinados com um conjunto de dados, mas através de tentativa e erro. Assim os algoritmos recebem *feedback* positivo quando se estão a aproximar do objetivo ou negativo quando se afastam, resultando numa aprendizagem semelhante àquela presente nos seres vivos [32].

Este tipo de aprendizagem é muito utilizada em jogos ou em robôs. Um exemplo seria um jogo em que existiria um labirinto, um objetivo e inimigos que eliminam o jogador. O algoritmo iria tentar percorrer o labirinto e cada vez que se aproximava do objetivo seria recompensado, no entanto se fosse de encontro a um inimigo iria receber uma penalização de modo que iria aprendendo a evitar os inimigos e a chegar ao objetivo através do melhor caminho.

Podemos pensar que aprendizagem por reforço é um pouco como treinar um cão recorrendo a biscoitos. Se os cães tiverem um biscoito por cada vez que obedecerem a uma ordem eventualmente irão perceber que é esse o comportamento pretendido e irão realizar essa ação todas as vezes [32].

2.2.3.1 *Q Learning*

O *Q-learning* é um algoritmo de aprendizagem que utiliza *Temporal Difference Learning*. Este método de aprendizagem por reforço começa por assumir que todos os estados e ações apresentam um valor inicial 0, e com o decorrer do tempo, vai atualizar os valores através do cálculo da diferença entre o valor esperado com o valor encontrado. O *Q-learning* segue a seguinte fórmula para calcular os seus novos valores:

$$novoQ(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- $Q(s_t, a_t)$: valor de escolher uma ação a_t num estado s_t;
- r_{t+1}: recompensa imediata;
- α: taxa de aprendizagem, $0 < \alpha << 1$. Proporção usada para atualizar o valor de utilidade após cada ação;
- γ: fator de desconto, $0 << \gamma < 1$. Encoraja o agente a preferir recompensas imediatas a tardias;

Exemplificando o comportamento de um agente no estado 1, este começa por realizar a ação 1 e recebe a recompensa 1. Em seguida vê a recompensa máxima no estado 2 e atualiza o valor da ação 1 realizada nesse estado. Vai posteriormente seguindo sempre esta metodologia.

2.2.3.2 *SARSA*

O SARSA é um algoritmo alternativo ao *Q-learning* que utiliza também *Temporal Difference Learning* como método de aprendizagem automática. A fórmula do SARSA pode ser representada da seguinte forma:

$$novoQ(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- $Q(s_t, a_t)$: valor de escolher uma ação a_t num estado s_t;
- r_{t+1}: recompensa imediata;
- α: taxa de aprendizagem, $0 < \alpha << 1$. Proporção usada para atualizar o valor de utilidade após cada ação;
- γ: fator de desconto, $0 << \gamma < 1$. Encoraja o agente a preferir recompensas imediatas a tardias;

Podemos verificar que a fórmula é muito semelhante, resultando num comportamento ligeiramente diferente por parte do agente. Um agente no estado 1 realiza a ação 1 e recebe a recompensa 1. No estado 2, realiza a ação 2 e obtém a recompensa 2 e em seguida atualiza o valor da ação 1 no estado 1. Segue então depois esta metodologia até terminar.

2.3 Literatura científica

Com o aumento exponencial dos sensores facilmente acessíveis a todos, maioritariamente a partir dos nossos *smartphones*, também cresceu imenso a tentativa de reconhecer atividades do ser humano através dos sensores nestes presentes. Existe no entanto um sensor que se destaca claramente dos outros, o acelerómetro. Este sensor como já vimos anteriormente fornece dados muito interessantes para deteção de movimentos, para além de também marcar uma presença muito assídua nos telemóveis que usamos no dia a dia. Em seguida iremos analisar alguns estudos nesta área e perceber como os investigadores procederam para realizar o seu reconhecimento de atividades.

2.3.1 *Activity Recognition with Smartphone Sensors*

Neste primeiro artigo [39] é realizado um estudo global relativamente ao reconhecimento de atividades através de sensores de *smartphones*. Assim é realizada uma revisão relativamente a várias técnicas de coleção de dados, analisados os maiores desafios e ainda mostradas algumas aplicações que são possíveis graças a este tipo de reconhecimento.

Os autores primeiramente destacam um conjunto de sensores que costumam ser usados no reconhecimento de atividades. Esse conjunto é então formado pelos seguintes sensores:

- Acelerómetro;
- Sensor de temperatura ambiente;
- Sensor de gravidade;
- Giroscópio;
- Sensor de luminosidade;
- Sensor de aceleração linear;
- Magnemómetro;
- Barómetro;
- Sensor de proximidade;

- Sensor de humidade.

Relativamente às atividades a serem reconhecidas, estas costumam subdividir-se em 5 categorias, atividades simples, atividades complexas, atividades do dia a dia, atividades de trabalho e ainda atividades relacionadas com a saúde. Na tabela 2 pode-se observar com mais detalhe as atividades relativas a cada categoria.

Categoria	Atividades
Atividades simples	Caminhar, correr, sentar, em pé, deitado, subir escadas, descer escadas, saltar, subir escadas rolantes, descer escadas rolantes, subir de elevador, descer de elevador.
Atividades complexas	Ir às compras, apanhar autocarros, andar a pé, conduzir um carro.
Atividades do dia a dia	Escovar os dentes, aspirar, escrever num teclado, comer, cozinhar, lavar as mãos, meditação, bater palmas, regar plantas, varrer, fazer a barba, secar o cabelo, lavar a louça, passar a ferro, fazer uma descarga na sanita.
Atividades de trabalho	Trabalhar, relaxar, limpar, numa pausa, reunião, conversação em casa, entretenimento em casa.
Atividades para a saúde	Exercício, quedas, atividades de reabilitação, Seguir rotinas.

Tabela 2: Tipos de atividades por categoria

Os dados e a sua maneira de recolha são pontos fulcrais para a qualidade final do reconhecimento das atividades. Deste modo, é preciso ter em atenção não só ao número de sensores utilizados, como também a sua variabilidade. Além disso, é também necessário estabelecer um equilíbrio em relação à taxa de recolha de dados, uma vez que, embora uma taxa mais elevada forneça mais informação, também adiciona consideravelmente mais ruído.

Após recolhidos os dados será então depois necessário proceder ao seu pré-processamento com o objetivo de reduzir o seu ruído. Este ruído pode ser provocado quer pelos utilizadores, quer pelos próprios sensores. Um segundo passo de pré-processamento importante é a segmentação dos dados, isto é, dividir dados contínuos em pequenos segmentos a partir dos quais é possível retirar *features* e poder treinar modelos.

No seguinte passo será então preciso estabelecer as *features* necessárias para o treino do modelo. Para reconhecimento de atividades as *features* estão subdividas em *features* do domínio do tempo e do domínio da frequência, que podem ser vistas com mais detalhe na tabela 3.

Domínio	Features
Tempo	Médias, máximos, mínimos, desvios padrões, variância, correlação, área sinal-magnitude (SMA).
Frequência	Energia, entropia, tempo entre máximos, distribuições.

Tabela 3: Tipos de *features* por domínio

Visto o reconhecimento de atividades ser um problema de classificação multi-classe, muitos classificadores podem ser usados para treinar os modelos. Desta forma os modelos destacados pelos autores foram os seguintes:

- Árvores de decisão;
- *Weka Toolkit*;
- Tabelas de decisão;
- KNN;
- HMM;
- SVM;
- Redes neuronais.

Para além destes classificadores base, podem ainda ser aplicados classificadores que agrupam vários modelos. Estes estão sub-divididos em três categorias, *voting*, *stacking* e *cascading* (mais detalhe na tabela 4).

Categoria	Descrição
Voting	Cada classificador base fornece um voto para a classifcação final.
Stacking	Um algoritmo de aprendizagem é utilizado para aprender como combinar as previsões dos classficadores base.
Cascading	É usado um mecanismo dividido em várias fases. O resultado de um dado classficador é colecioando como informação adicional para o classificador seguinte.

Tabela 4: Descrição das categorias de utillização de múltiplos classificadores

Neste artigo os autores não realizaram qualquer previsão de atividades, focando-se apenas nos princípios fundamentais dos passos que são habitualmente tomados em estudos que tentam realizar a deteção de atividades utilizando sensores de telemóveis.

2.3.2 *A Study on Human Activity Recognition Using Accelerometer Data from Smartphones*

No artigo [40] os seus autores realizaram um sistema capaz de reconhecer atividades humanas usando apenas dados de acelerómetros incorporados em *smartphones*.

Os dados colecionados foram provenientes de acelerómetros de três eixos presentes em telemóveis *Android*. Foram recolhidos dados por quatro pessoas, duas de cada género, com idades compreendidas entre 29 e 33 anos. Neste estudo todos os participantes realizaram as seis atividades a reconhecer, corrida, caminhada lenta, caminhada rápida, dança aeróbica, subir escadas, e descer escadas. A taxa de amostragem do estudo foi de 10ms sendo que cada período da atividade durou entre 180 e 280 segundos. Na experiência aquando a recolha de dados o telemóvel não poderia ir da mão para o bolso nem vice-versa uma vez que os dados da aceleração são muito sensíveis a estas mudanças.

Na extração de *features* foi aplicada uma técnica denominada *window overlapping*. Assim, os autores dividiram os dados em subconjuntos de 128 amostras, o que corresponde a 1.28 segundos de dados. Foram escolhidas as *features* mais relevantes totalizando um número de 18 *features*:

- Média aceleremómetro ao longo do eixo z;
- Diferença entre o máximo e o mínimo da magnitude da aceleração;
- Desvio padrão da magnitude da aceleração;
- Raiz quadrada média da magnitude da aceleração;
- Número médio de picos em cada eixo;
- Variância entre o número médio dos picos;
- Desvio padrão em cada eixo;
- Raiz quadrada média em eixo;
- Correlação entre o eixo z e y;
- Diferença entre o máximo e o mínimo ao longo de cada eixo.

Para a classificação foram dividas os dados recolhidos com o telemóvel no bolso daqueles em que o telemóvel se encontrava na mão dos participantes. Primeiramente foram testados os classificadores *multilayer perceptron*, SVM, florestas aleatórias, LMT, regressão logística e *logit boost*, sendo que a primeira obteve os melhores resultados de *accuracy* quer para os dados na mão (89.48%), quer para os dados no bolso (89.72%). Para tentar melhorar os resultados foi usada a técnica de fusão de probabilidades médias, obtendo como melhor resultado de *accuracy* para dados recolhidos na mão a combinação MLP,

logit boost e SVM (91.15%) e para os dados recolhidos no bolso a combinação MLP, florestas aleatórias, regressão logística (90.34%).

2.3.3 *A framework for group activity detection and recognition using smartphone sensors and beacons*

No estudo presente no artigo [41] os autores criaram uma *framework* para detetar atividades em grupo, mais concretamente em aula, numa discussão ou num seminário. Para além disso é também possível detetar se um utilizador se encontra a realizar uma atividade individual. Para isso foram utilizados dados captados por acelerómetros, microfones e ainda por *beacons*.

De modo a colecionar os dados, os autores criaram uma aplicação *Android* capaz de recolher os dados dos sensores já referidos. Todos os dados foram recolhidos com os telemóveis colocados nos bolsos dos participantes e as atividades recolhidas apresentavam durações de 330 segundos.

Baseando-se na *feature* da aceleração, a *framework* tem em atenção a sua média, máximo, mínimo, variância, raiz quadrada média, amplitude interquartil, energia espectral e entropia do sinal. Para a *feature* do áudio foi utilizada uma estratégia que usando um valor fixo dos dados de fala, determina momentos de fala ou silêncio. Por fim para a *feature* da localização, os dados dos *beacons* das redondezas são captados para determinar a localização do dispositivo assim como se existem outros dispositivos por perto. Estas *features* foram depois combinadas e determinados os seus pesos de modo a refletirem a importância da contribuição de cada uma para o resultado final.

O classificador com maior média *accuracy* (90.09%) e menor variância de *accuracy* (0.024) foi a árvore de decisão. Para além deste classificador foram também testados *k-nearest neighbors*, rede neuronal profunda, processo gaussiano, regressão logística, SVM, análise descriminante linear e *naive bayes*.

2.3.4 *Detecting User Activities with the Accelerometer on Android Smartphones*

A deteção de atividades utilizando apenas o acelerómetro incorporado em *smartphones* voltou a ser uma abordagem a ser seguida no artigo [42]. As atividades a serem reconhecidas são telemóvel separado do utilizador, repouso, andar, correr e saltar.

As atividades já identificadas foram realizadas diariamente por 10 utilizadores distintos. Para os dados serem recolhidos, os utilizadores faziam-se acompanhar de telemóveis *Android* no seu bolso frontal das calças. Os dados coletados pela aplicação criada pelos autores colecionava dados a cada 20 ms, sendo que a duração de cada atividade era de cerca de 30 segundos.

Na geração de *features* foi utilizada uma *sliding window* de 100 amostras. No que toca às *features* a usar para treinar o modelo, os autores optaram pelas seguintes:

- Frequência fundamental do sinal;
- Amplitude máxima do sinal;
- Amplitude mínima do sinal;
- Intensidade do valor do sinal;
- Magnitude das frequências fundamentais;
- Número de passos;
- Direção do posicionamento do telemóvel.

Para a classificação do modelo foi utilizada a classificação *k-nearest neigbor* dos dados. Os autores não apresentaram quaisquer valor de performance do modelo, apresentando apenas uma tabela com os casos de teste assim como o seu sucesso ou insucesso na deteção. Segunda esta tabela, o modelo mostrou-se capaz de detetar as atividades de telemóvel separado do utilizador, repouso, andar e saltar. No entanto para os casos de correr o modelo tendia a confundi-los com a atividade de saltar.

2.3.5 *Activity Detection and Analysis Using Smartphone Sensors*

Uma outra perspetiva para reconhecimento de atividades surgiu no artigo [4]. Estes autores utilizaram o acelerómetro e o giroscópio para tentar prever se o utilizador estava parado em pé, a andar, a correr ou se o dispositivo estava estacionário.

Para recolher os dados os autores criaram uma aplicação *Android*. Esta aplicação foi utilizada por apenas um utilizador de 23 anos, num telemóvel Samsung Galaxy S4. Os testes foram corridos para cada atividade quatro vezes, cada uma com o telemóvel numa orientação diferente. Durante a recolha era pedido ao participante que andasse a um ritmo normal por 25 segundos, depois ficasse 25 segundos de pé e por fim que corressem em linha reta outros 25 segundos. Para além destes dados foram também colecionados dados em que o telemóvel se encontrava estacionário em cima de uma mesa.

As *features* utilizadas no estudo não foram reveladas, no entanto é referido que os modelos foram treinados com recurso ao *Weka* uma *workbench* de *machine learning* que gera diversos modelos. Os algoritmos usados foram o *k-means*, *naive bayes*, arvore de decisão J48 e florestas aleatórias.

No final o classificador com melhores resultados foi o das florestas aleatórias com uma *accuracy* de 89.56%. Os autores disseram ainda que como trabalho futuro pretendiam utilizar uma rede neuronal pois pensam que poderá melhorar o resultado final da deteção.

Capítulo 3

Activity Tracker - Aplicação de Recolha de Dados

Este capítulo irá-se debruçar sobre a aplicação de recolha de dados. Será apresentada a forma usada para recolher, armazenar e estruturar os dados de forma a poderem depois ser usados na fase seguinte.

Para qualquer projeto de inteligência artificial, como pudemos ver no capítulo anterior, são necessários dados para treinar os modelos. Assim é necessário arranjá-los, e sendo que não é possível encontrar os dados necessários para este projeto a partir de uma fonte externa, foi então necessário criar algo para os recolher. Para além disso também foi preciso rotulá-los relativamente à atividade a ser realizada. Esta necessidade resultou na criação do *Activity Tracker* (figura 16).

Figura 16: *Activity Tracker - icon* da aplicação

3.1 Aplicação *Android*

Sendo o objetivo do projeto a identificação de atividades recorrendo a *smartphones*, só faria sentido o dispositivo de recolha dos dados ser um *smartphone*. Deste modo foi criada uma aplicação *android* capaz de recolher dados dos seguintes sensores do telemóvel:

- Acelerómetro;
- Giroscópio;
- Sensor de gravidade;
- Sensor de orientação;
- Sensor de luminosidade;
- Sensor de proximidade;
- Bluetooth;
- Conectividade.

Em adição a estes dados foi ainda recolhido o identificador do *smartphone* para permitir identificar dispositivos que forneçam constantemente dados errados.

A aplicação denominada de *Activity Tracker* tira proveito da documentação do *android sensors* [16] para o desenvolvimento mais ágil das funcionalidades. A aplicação é formada por duas *Activities* do *android*, uma para o utilizador escolher a atividade que vai realizar e uma outra para o utilizador poder controlar a recolha dos dados. As atividades que se pretendem detetar são estar parado, andar, correr, andar de bicicleta e conduzir, daí as opções apresentadas no menu representado na figura 17.

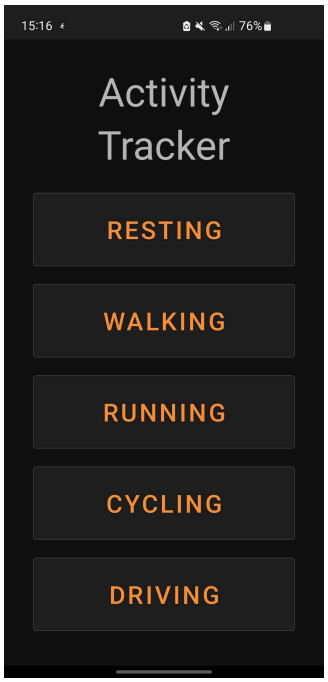


Figura 17: *Activity Tracker* - menu seleção de atividade

Depois de selecionar uma atividade, o utilizador passa para a página da figura 18 e pode controlar a recolha dos dados utilizando os seguintes botões:

- ***Start*** - começa/retoma a recolha de dados;
- ***Stop*** - para a recolha de dados;
- ***Save Data*** - guarda os dados da sessão na base de dados;
- ***Delete Data*** - elimina os dados recolhidos na presente sessão.

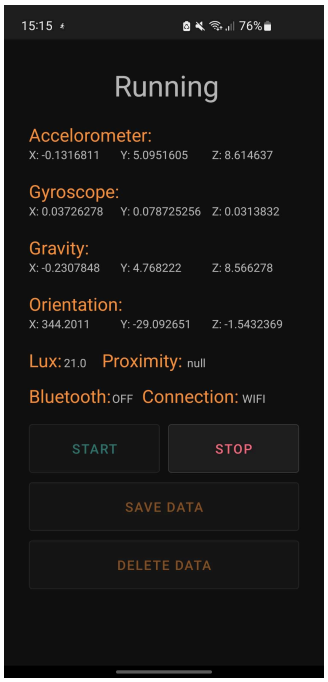


Figura 18: *Activity Tracker* - menu seleção de atividade

A aplicação a cada segundo visualiza os dados dos seus sensores e acrescenta um novo objeto de dados à sua lista de recolha, que no final da execução da atividade, com o recurso do botão *Save Data* envia para a base de dados o resultado da atividade, contendo a lista de todos os dados recolhidos.

Uma vez os dados dos sensores serem considerados sensíveis, a *google* não permite a sua recolha enquanto o utilizador está com o ecrã do telemóvel desligado, de modo que foi necessário criar um serviço *foreground* do *android* para informar o utilizador que a aplicação se encontra em execução em segundo plano e assim poder recolher dados desde que a notificação enviada esteja ativa.

3.2 Servidor

Para poder guardar os dados resultantes do uso da aplicação foi necessário criar um servidor capaz de comunicar com a aplicação e guardar os dados numa base de dados. Para isso foi necessário criar uma aplicação e colocá-la sempre disponível para os utilizadores poderem recolher dados quando quiserem. Assim uma máquina virtual na *google cloud platform* foi criada e instalada a aplicação. Para a construção

desta foi utilizado o *Quarkus*, que é uma *framework* em *java* utilizada para construir a aplicações capazes de comunicar com outras a partir de REST APIs. Foi escolhida esta *framework* por neste momento ser considerada uma das melhores *frameworks* para *java*, sendo melhor em muitos aspetos relativamente ao *standard* atual, o *Spring* [43]. No desenvolvimento foram seguidos os padrões do *Quarkus* obtendo uma divisão em 4 partes, *entities*, *repositories*, *services* e *resources*.

3.2.1 *Entities*

Nesta secção é onde são definidas as classes persistentes, isto é, aquelas que serão guardadas na base de dados. Assim aqui existem duas classes, *Activity* e *TimeStamp*. Para a *framework* perceber que estas são as classes a persistir é necessário primeiro adicionar uma anotação *@Entity* à classe. Em seguida é necessário declarar uma variável que representará a identificação única de cada objeto, normalmente dada o nome de *id*. Em seguida podem ser definidas as outras variáveis do objeto como normalmente se faz em *java*. Por fim é possível adicionar as relações que serão criadas na base de dados através das anotações *@OneToMany, @OneToOne, @ManyToMany, @ManyToOne* que irão resultar nas relações expectáveis em *SQL*. Por fim é necessário criar pelo menos o construtor vazio e realizar os normais *getters* e *setters* para todas as variáveis. Na figura 19 é possível ver um código de exemplo para criar a entidade *Activity*.

```java
@Entity
public class Activity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", nullable = false)
    private Long id;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "activity_id")
    private List<TimeStamp> timeStamps;

    public Activity() {}

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public List<TimeStamp> getTimeStamps() { return timeStamps; }

    public void setTimeStamps(List<TimeStamp> timeStamps) {
        this.timeStamps = timeStamps;

    public void addTimeStamp(List<TimeStamp> timeStamps) {
        this.timeStamps.addAll(timeStamps
    }
}
```

Figura 19: Exemplo de código de uma *entity*

3.2.2 *Repositories*

Existe um *repository* para cada *entity* a ser persistida. Estas classes são aquelas que permitem interagir com a base de dados, quer seja para guardar, alterar ou ler dados. Para realizar esta interação foi escolhido

o *PanacheRepository* como mapeador relacional de objetos, que permite realizar todas estas operações. Este é uma simplificação do *Hibernate* ORM que é uma *framework* para interligar aplicações a base de dados de uma forma mais direta e simples. Dentro desta classe podem ser realizados métodos para fazer qualquer pesquisa na base de dados. Na figura 20 pode-se ver um exemplo de uma classe deste tipo.

```java
@ApplicationScoped
public class ActivityRepository implements PanacheRepository<Activity> {

    public Activity findActivityById(Long id) {
        return find("id", id).firstResult();
    }
}
```

Figura 20: Exemplo de código de um *repository*

3.2.3 *Services*

As classes aqui presentem servem para realizar toda a lógica da aplicação. Um *service* pega num ou mais *repositories* para poder aceder e guardar os dados. Um método normal desta classe tem a ir buscar informação aos *repositories*, depois modifica-os ou realiza algum tipo de calculo com estes dados e depois volta a guardá-los recorrendo aos mesmos. Todos os *repositories* presentes nestas classes precisam de ter a anotação *@Inject* para funcionarem corretamente. Para além disso é também necessário incluir a anotação *@Transactional* em todos os métodos que realizem alterações na base de dados. Assim na figura 21 pode ver-se um exemplo de um serviço.

```java
@ApplicationScoped
public class ActivityService {

    private static final ObjectMapper OBJECT_MAPPER =
JsonMapper.builder().findAndAddModules().build();
    @Inject
    ActivityRepository activityRepository;

    @Transactional
    public void deleteActivity(long id) {
        Activity activity = activityRepository.findActivityById(id);
        activityRepository.delete(activity);
    }
}
```

Figura 21: Exemplo de código de um *service*

3.2.3.1 *Resources*

Um *resource* é o que permite a comunicação da aplicação com o exterior, nomeadamente a aplicação *android* desenvolvida. O servidor onde está localizada a aplicação possui um endereço fixo. Com a adição de alguns campos ao caminho inicial podem ser executadas chamadas à aplicação e assim enviar ou receber dados através de uma REST API. Um *resource* primeiramente para funcionar corretamente precisa

de ter na sua classe a anotação *@Path* para indicar o caminho base para as chamadas à API. Em seguida cada método terá de apresentar esta mesma anotação, para indicar o seu caminho mais específico e ainda conter uma anotação do tipo de chamada, sendo as mais usadas *@GET* e *@POST*, usadas segundo a norma, para receber ou enviar dados respetivamente. Estes métodos não possuem qualquer lógica da aplicação, portanto precisam de ter *services* com a anotação *@Inject* para poderem executar algum tipo ação. Na figura 22 pode-se ver um exemplo de *resource*.

```java
@Path("/activity")
public class ActivityResource {
    @Inject
    ActivityService activityService;

    @POST
    @Consumes("application/json")
    @Path("/add")
    public long addActivity(String request) throws JsonProcessingException {
        return activityService.addActivity(request);
    }

    @POST
    @Path("/delete/{id}")
    public void deleteActivity(@PathParam("id") long id) {
        activityService.deleteActivity(id);
    }
}
```

Figura 22: Exemplo de código de um *resource*

3.2.4 Caso de exemplo

Imaginado que o endereço do servidor é `https://www.app.com` para eliminar uma atividade com o *id* 245 através do código fornecido nas figuras 19 a 22, começa-se por realizar uma chamada para o endereço `https://www.app.com/activity/delete/245` utilizando o tipo *post*. Esta chamada corresponderá à chamada do método *deleteActivity* no *resource*. Este por sua vez chama o método *deleteActivity* do *service*. Em seguida este tira proveito do método *findActivityById* do *repository* para obter a *entity Activity* e elimina essa mesma *entity* através do uso do mesmo *repository* através do método *delete*.

3.3 Base de Dados

3.3.1 Estrutura dos dados

Para armazenar dados foi construída uma base de dados relacional utilizando o *PostgreSQL* com a estrutura representada na figura 23.

A estrutura da base de dados foi desenhada desta forma simplesmente para facilitar o próximo passo do projeto, o tratamento de dados. Habitualmente esta fase utiliza ficheiros no formato CSV que com esta estrutura da base de dados é possível realizar uma transformação transparente da tabela *timestamp* para este formato.

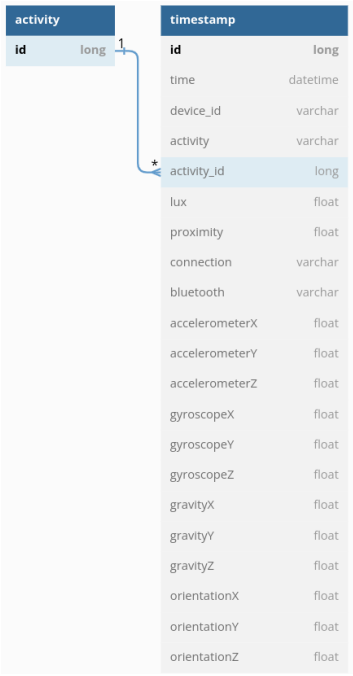


Figura 23: Estrutura da base de dados

3.3.2 Dados resultantes

Quando se extrai os dados presentes na base de dados, obtém-se um ficheiro CSV constituídos pelas colunas correspondentes a cada um dos valores dos sensores recolhidos com a adição do momento da recolha dos dados, o identificador do dispositivo, o identificador da atividade e ainda o identificador da linha correspondente. Um excerto pode ser observado na figura 24.

id	accelerometer	accelerometer	accelerom	activity	bluetooth	connectio	deviceid	gravityx	gravityy	gravityz	gyroscopex	gyroscopey	gyroscopez	lux	orientatio	orientationy	orientation	proximity	time	activity_id
462	-2.60295	6.6979000	5.5150500	Resting	OFF	WIFI	32cb6804d	-1.85235	6.711789	6.905871	-0.410025	-0.031625	-0.117975	40	320.92188	-43.1875	-10.890625	5	2022-11-0	461
463	0.81700000	7.04505	7.122	Resting	OFF	WIFI	32cb6804d	-1.55288	6.346114	7.313398	0.0650375	-0.1530375	-0.3001625	40	321.125	-40.328125	-9.109375	5	2022-11-0	461
464	-5.15805	2.76705	9.84405	Resting	OFF	WIFI	32cb6804d	-1.44719	5.375552	8.07338	-0.132	-0.5418875	-0.7209125	42	325.73438	-33.234375	-8.484375	5	2022-11-0	461
465	-1.092	5.6899505	7.6420503	Resting	OFF	WIFI	32cb6804d	-0.997642	5.791906	7.850409	0.0554125	0.0089375	-0.016775	41	316.625	-36.203125	-5.84375	5	2022-11-0	461
466	-1.0900501	7.1670003	6.5670004	Resting	OFF	WIFI	32cb6804d	-0.825103	7.367132	6.419887	-0.0862100	-0.0103124000	-0.032725	40	314.73438	-48.703125	-4.828125	5	2022-11-0	461
467	-0.63000000	7.0330505	6.45195	Resting	OFF	WIFI	32cb6804d	-0.737558	7.058876	6.767472	0.0235125	-0.0166375	0.0077	38	314.53125	-46.046875	-4.3125	5	2022-11-0	461
468	-0.642	6.97305	6.543	Resting	OFF	WIFI	32cb6804d	-0.684121	7.031969	6.801013	0.0149875	-0.0070125	-0.0028875	37	314.6875	-45.8125	-4	5	2022-11-0	461
469	-0.90105003	6.7120504	6.9900002	Resting	OFF	WIFI	32cb6804d	-0.743358	6.843458	6.984618	-0.0308	-0.0166375	-0.0380875	38	315.84375	-44.25	-4.34375	5	2022-11-0	461
470	-0.55605	6.0349503	8.08005	Resting	OFF	WIFI	32cb6804d	-0.594598	6.380623	7.42324	-0.0287375	0.0738375	-0.0050875	41	321.17188	-40.593754	-3.46875	5	2022-11-0	461
471	-0.55905	6.11595	7.52895	Resting	OFF	WIFI	32cb6804d	-0.511139	6.238293	7.54936	-0.006325	-0.011275	0.000275	41	320.125	-39.5	-2.984375	5	2022-11-0	461
472	-0.50100005	6.1710005	7.3879504	Resting	OFF	WIFI	32cb6804d	-0.529151	6.206941	7.57392	0.010725	0.0004125	-0.00605	42	319.3125	-39.265625	-3.09375	5	2022-11-0	461
541	0.05746084	0.079008654	9.809343	Resting	OFF	WIFI	f38c1b511	0.07232800	0.050809875	9.8062525	0.00045814895	-0.0012217305	0.00045814895	8	297.6163	-0.29686025	0.4225916	5	2022-11-1	540
542	0.14604631	0.01436521	9.900323	Resting	OFF	WIFI	f38c1b511	0.1391353	0.033197805	9.805607	0.0053450707	0.018325957	-0.00015271600	20	297.594	-0.19396001	0.812936	5	2022-11-1	540
543	0.10295067	0.05267244	9.890746	Resting	OFF	WIFI	f38c1b511	0.08335838	0.049310096	9.806171	0.0010690142	0.0030543262	-0.00076358154	21	297.7561	-0.2880976	0.487037	5	2022-11-1	540
544	0.11971009	0.06464344	9.907506	Resting	OFF	WIFI	f38c1b511	0.080099040	0.053986773	9.806174	-0.001985312	0.0030543262	-0.00076358154	27	297.769	-0.31542167	0.46799448	5	2022-11-1	540
545	0.110133275	0.035913024	9.883564	Resting	OFF	WIFI	f38c1b511	0.053589925	0.04376395	9.806406	0.00045814895	-0.002443461	0.00045814895	27	297.79773	-0.25569364	0.31310615	5	2022-11-1	540
546	0.009576807	0.02873042	9.933842	Resting	OFF	WIFI	f38c1b511	0.07040147	0.047051396	9.806285	0.0010690142	-0.0036651916	-0.00015271631	30	297.81583	-0.27490088	0.41133186	5	2022-11-1	540
547	0.055066638	0.04070143	9.890746	Resting	OFF	WIFI	f38c1b511	0.05391208	0.04328234	9.806406	-0.00015271630	0	-0.00015271631	25	297.8411	-0.25287977	0.31498832	5	2022-11-1	540
548	0.062249243	0.023942016	9.890746	Resting	OFF	WIFI	f38c1b511	0.057228602	0.043878928	9.806385	0.0010690142	-0.002443461	-0.00015271631	27	297.87045	-0.2563654	0.33436584	5	2022-11-1	540
549	0.05267244	0.059855044	9.902718	Resting	OFF	WIFI	f38c1b511	0.053164996	0.04859788	9.806385	0.00045814895	-0.0012217305	-0.00015271631	26	297.89075	-0.2839364	0.31062415	5	2022-11-1	540
550	0.047884032	0.059855044	9.890746	Resting	OFF	WIFI	f38c1b511	0.056678925	0.046967845	9.806374	0.0008399397	-0.0009162979	0.00015271631	25	297.91156	-0.27441272	0.33115473	5	2022-11-1	540
551	0.06703765	0.047884032	9.8955345	Resting	OFF	WIFI	f38c1b511	0.05499842	0.042853747	9.806402	0.0008399397	-0.0009162979	-0.00045814895	25	297.91986	-0.2503757	0.3213354	5	2022-11-1	540

Figura 24: Dados resultantes

As colunas presentes no ficheiro representam o seguinte:

- ***id*:** identificador da linha de dados;
- ***accelerometerx*:** dados do valor x do acelerómetro;
- ***accelerometery*:** dados do valor y do acelerómetro;

- ***accelerometerz*:** dados do valor z do acelerómetro;
- ***activity*:** atividade a ser realizada;
- ***bluetooth*:** estado do *bluetooth*;
- ***connection*:** estado da *conexão*;
- ***deviceid*:** identificador do dispositivo;
- ***gravityx*:** dados do valor x do sensor de gravidade;
- ***gravityy*:** dados do valor y do sensor de gravidade;
- ***gravityz*:** dados do valor z do sensor de gravidade;
- ***gyroscopex*:** dados do valor x do giroscópio;
- ***gyroscopey*:** dados do valor y do giroscópio;
- ***gyroscopez*:** dados do valor z do giroscópio;
- ***lux*:** valor do sensor de luminosidade;
- ***orientationx*:** dados do valor x do sensor de orientação;
- ***orientationy*:** dados do valor y do sensor de orientação;
- ***orientationz*:** dados do valor z do sensor de orientação;
- ***proximity*:** valor do sensor de proximidade;
- ***time*:** momento de recolha;
- ***activity_id*:** identificador da sessão.

Como se pode verificar olhando para estes dados, a sua utilidade atual não é muito grande, uma vez que para tirar realmente proveito dos mesmos é necessário proceder ao seu processamento. Valores soltos de alguns destes sensores não dizem nada de relevante, no entanto a variação dos mesmos já se torna algo importante.

Desta forma será necessário em seguida processar os dados de forma a fazerem sentido e assim poderem posteriormente serem passados a modelos de *machine learning*.

Capítulo 4

Configuração Experimental

Este capítulo irá-se debruçar sobre a configuração experimental para a utilização de modelos de *machine learning*. Primeiro será mostrado como foi criada a aplicação de previsão assim como ela encaixa na aplicação *android* já criada. Em seguida será apresentada a forma como os dados foram preparados e as experiências realizada para obter os hiperparâmetros ótimos dos modelos desenvolvidos. Por fim é apresentada uma tabela final com as *features* utilizadas nos modelos finais, assim como a sua descrição.

4.1 Metodologia

Este estudo é essencialmente constituído por duas partes, a aplicação de recolha de dados, já anteriormente referida assim como uma aplicação capaz de realizar previsões de atividades que será explicada ao longo do presente capítulo.

4.1.1 Aplicação de previsão

De modo a ser possível realizar novas previsões com o modelo final obtido, foi necessário criar algo capaz de comunicar com a aplicação de recolha de dados e por sua vez aplicar o modelo treinado de forma a obter uma previsão. Assim, visto que todo o desenvolvimento de modelos e processamento dos dados ter sido realizado com a linguagem *Python*, foi utilizada a *framework Flask* para desenvolver esta aplicação. Esta *framework* para *Python* permite uma criação muito simples de uma REST API a partir de um único ficheiro, neste caso denominado de *main.py* como se pode ver na figura 25.

```python
import pickle
from flask import Flask, request, jsonify

model = pickle.load(open('model.sav', 'rb'))

def json_to_df (json_object):
    (...)

def preprocess (df):
    (...)

def predict(x):
    (...)

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        try:
            json_object = request.get_json()
            df = json_to_df(json_object)
            x_test = preprocess(df)
            result = predict(x_test)
            return jsonify({'prediction': str(result)})

        except Exception as e:
            return jsonify({'error': str(e)})

    return "OK"

if __name__ == '__main__':
    app.run()
```

Figura 25: Exemplo de código da aplicação *Flask*

Neste exemplo podemos ver que a aplicação é constituída por um modelo treinado (*model*) e por 4 funções distintas, 3 das quais auxiliares (*json_to_df*, *preprocess* e *predict*) e uma principal que é utilizada para realizar chamadas à API, denominada de *index*. De uma forma simples, a função *index* recebe chamadas quer com método *GET* para fins de debug, quer métodos *POST* que neste caso são aquelas que contém no seu interior uma amostra de uma atividade em formato *json*. Esta amostra dentro da função *index* é primeiro passada a *dataframe* com recurso à função *json_to_df*. O resultado é passado à função *preprocess* que realiza o pré-processamento dos dados. Por fim, o resultado da operação anterior, é passada à função *predict* que realiza uma previsão da amostra em questão. Finalmente esta resposta é guardada em formato *json* recorrendo à função *jsonify* e devolvida como resposta à chamada *POST* inicial.

4.1.2 Arquitetura final

A aplicação ficou com a estrutura que podemos encontrar na figura 26. Um utilizador pode utilizar a aplicação *android* para dois fins, fornecer mais dados de atividades ou prever a atividade que se encontra a realizar.

O fluxo de ambas as funcionalidades começa num *smartphone android* que comunica com um servidor que se encontra instalado numa máquina da *Google Cloud Platform*. Em seguida, caso o pedido recebido seja para guardar dados, este vai guardá-los numa base de dados do *PostgreSQL* através de um *driver JDBC*. Por outro lado, se o pedido recebido tivesse a intenção de realizar uma previsão de uma atividade, este servidor estabeleceria uma comunicação via *HTTP* com a aplicação de previsão, que, por sua vez, responderia com o resultado que o servidor deve devolver para a aplicação *android*.

Todos os diferentes componentes dentro da máquina virtual da *Google Cloud Platform* encontram-se instalados em *containers* diferentes que podem por isso, ser alterados facilmente de uma forma transparente.

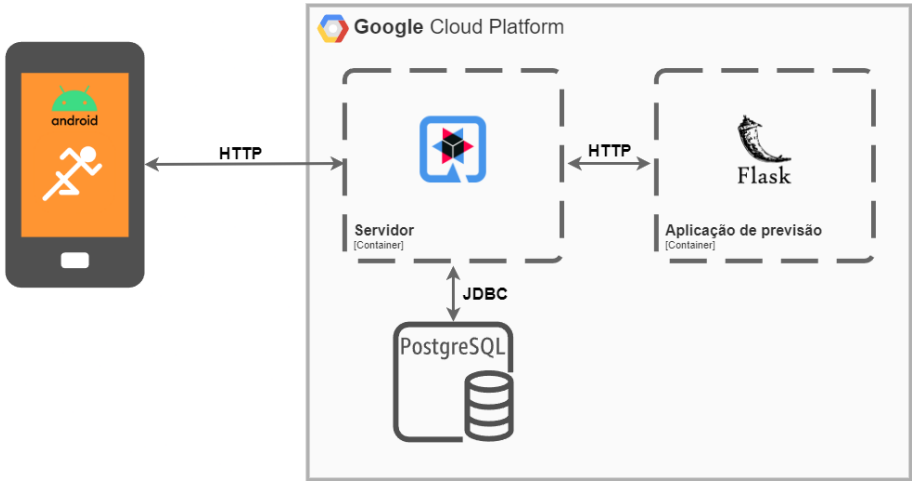


Figura 26: Arquitetura final da aplicação *Activity Tracker*

4.2 Preparação dos dados

Após recolhidos os dados, como pudemos ver anteriormente, obtemos valores soltos de medições de sensores. Estes valores em estado não processado por si só não representam nenhuma informação interessante. Deste modo foram seguidos os seguintes passos para obter dados relevantes para os modelos poderem ser treinados:

1. Dividir os dados por sessão de recolha;

2. No caso de valores numéricos calcular a diferença entre cada dois pontos consecutivos;

3. No caso de valores categóricos, a cada dois pontos de dados escolher o valor do primeiro ponto de dados;
4. Calcular novas *features* relevantes:
    - Aceleração linear, tendo por base dados do sensor de gravidade e do acelerómetro;
    - Magnitude da aceleração linear;
    - Magnitude da gravidade linear.
5. Converter dados categóricos em valores numéricos para os dados de tipo de conexão e estado *bluetooth*;
6. Agrupar os dados em amostras de 6 segundos e calcular os valores médios, máximos, mínimos e desvio padrão para valores numéricos;
7. Agrupar os dados em amostras de 6 segundos e escolher a moda para valores categóricos;
8. Remoção de colunas com valores de correlação superiores a 85%.

Após este processamento ficamos com diversas amostras de 6 segundos ligadas às diversas atividades, contendo *features* de valores médios, máximos, mínimos e desvios padrões das *features* numéricas mais relevantes, assim como a moda das *features* categóricas mais relevantes nesse período de tempo.

4.2.1 Balanceamento dos dados

Observando a figura 27 podemos ver uma quantidade heterogénea de amostras para cada tipo de atividade, por exemplo a atividade menos representada, *resting*, apresenta 9322 amostras enquanto que a mais representada, *driving* apresenta 50185 amostras.

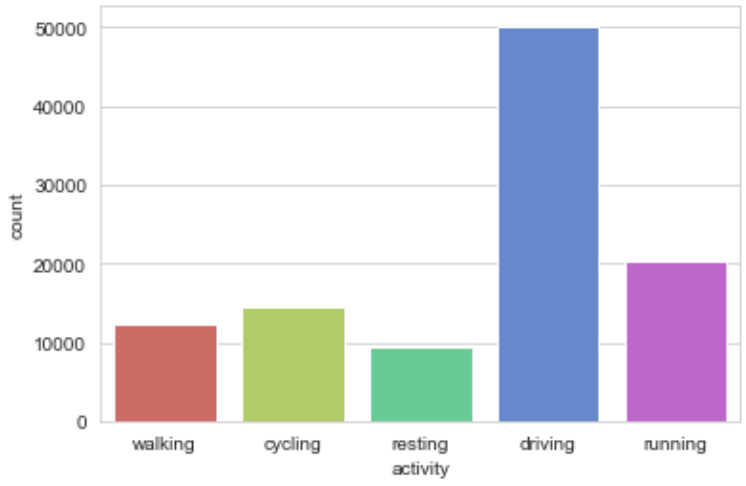


Figura 27: Número total de amostras de 6 segundos por atividade

Utilizar coleções de dados desbalanceados no treino de modelos, pode levar a que o modelo crie uma certa tendência para as classes mais representadas, isto é, aquando da previsão pode tender a sua escolha para a classe com maior volume de dados e assim por vezes não escolher a classe mais acertada.

Para resolver este problema procedeu-se à utilização da técnica de *undersampling*, isto é manter todos os dados da classe com menor quantidade de dados e reduzir todas as outras para conterem uma quantidade semelhante de dados. Neste caso em específico foram escolhidas 10 mil amostras aleatórias das restantes classes, obtendo a coleção de dados representada na figura 28.

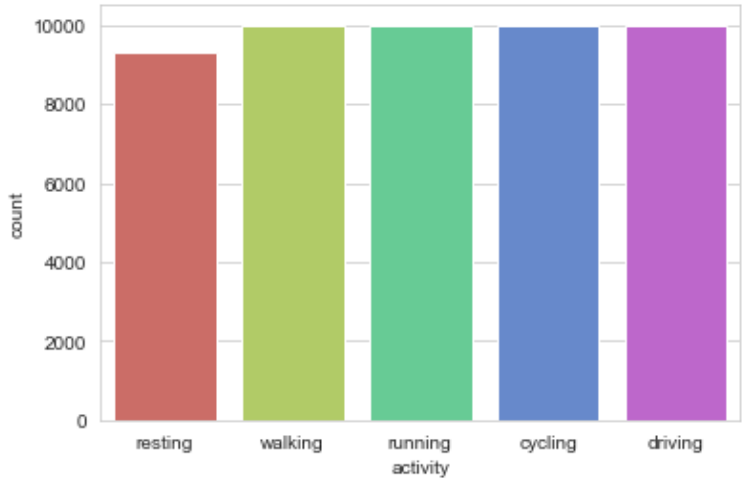


Figura 28: Número de amostras balanceadas de 6 segundos por atividade

4.2.2 Remoção de *outliers*

Uma vez os dados terem sido recolhidos em ambiente controlado, foi decidido não remover quaisquer amostras que tivessem valores possíveis de *outliers*. Esses dados, embora mais afastados dos valores médios, também representam a classe em questão, tal como qualquer outros, já que representam observações legítimas e portanto fazem naturalmente parte da população em estudo. Nas figuras 29, 30, 31 e 32 podemos observar os *boxplots* das *features* da média da aceleração no eixo do x, média da aceleração no eixo do y, média da aceleração no eixo do z e média da magnitude da aceleração linear respetivamente. Este tipo de gráficos permite a observação da distribuição dos dados relativamente a uma determinada *feature*, para todas as classes.

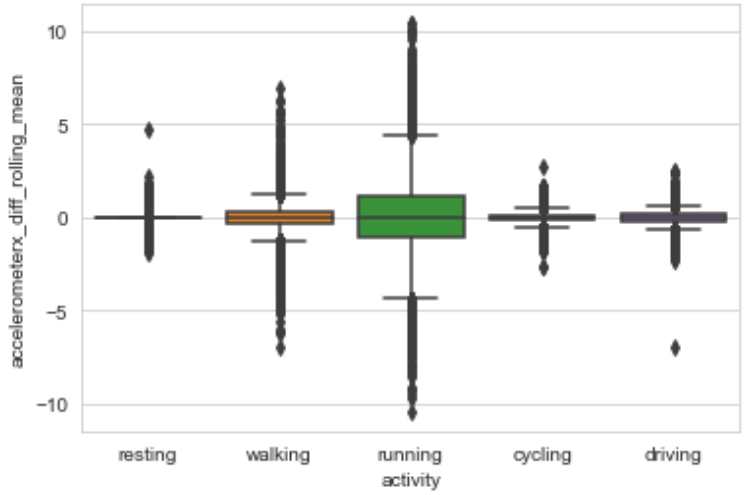


Figura 29: *Boxplot feature accelerometer_x_diff_rolling_mean*

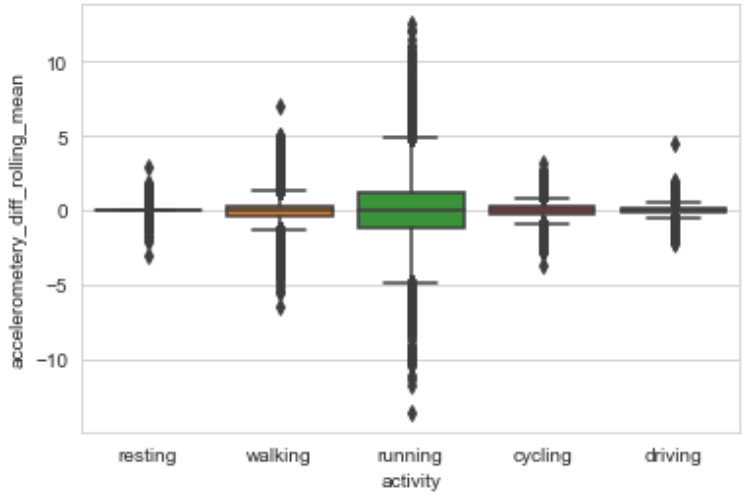


Figura 30: *Boxplot feature accelerometer_y_diff_rolling_mean*

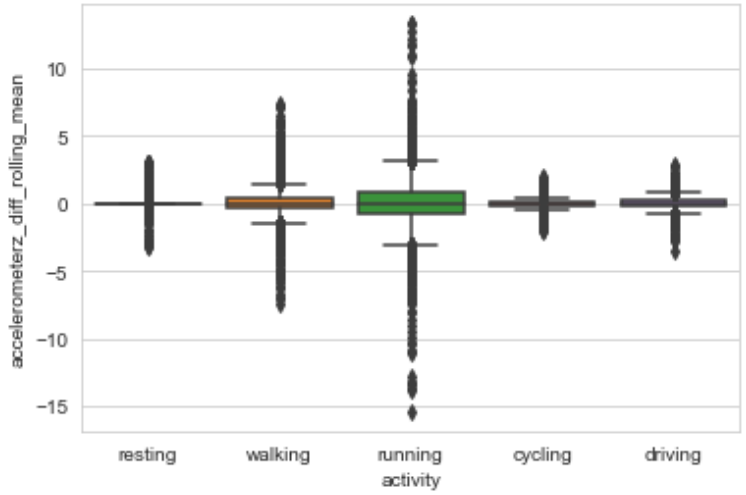


Figura 31: *Boxplot feature accelerometer_z_diff_rolling_mean*

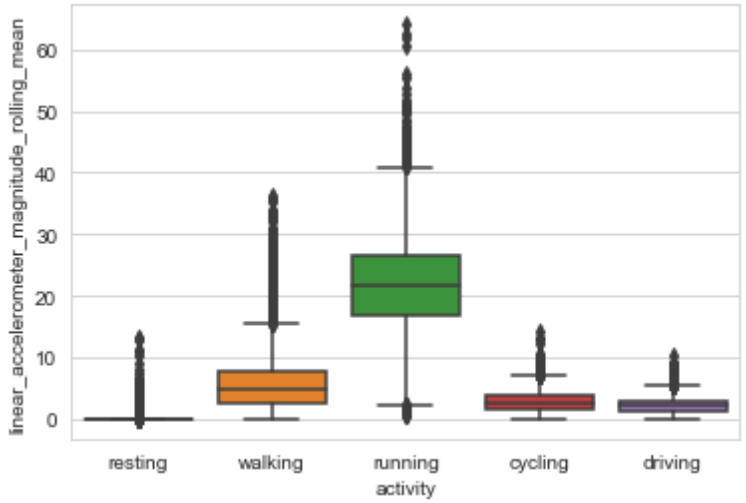


Figura 32: *Boxplot feature linear_accelerometer_magnitude_rolling_mean*

4.2.3 Remoção de *features* tendenciosas

Uma vez as amostras de dados recolhidas terem sido recolhidas por um grupo muito reduzido de pessoas, as amostras resultantes acabaram por revelar algumas *features* um pouco tendenciosas para cada uma das atividades, uma vez os dados serem recolhidos sempre na mesmas circunstâncias. Assim *features* relativamente ao sensor de *bluetooth* e sensor conectividade acabaram por ser removidas. Nas tabelas 5 e 6 podemos ver algumas dessas tendências dos valores de conectividade e *bluetooth*, respetivamente, para cada atividade.

Classe	Amostras com *bluetooth* ligado	Amostras com bluetooth ligado
repouso	660	9340
andar	5674	4326
correr	9755	245
andar de bicicleta	10000	0
andar de carro	8550	1450

Tabela 5: Quantidade de amostras cara cada estado de *bluetooth* por atividade

Classe	Amostras com WiFi ligado	Amostra com WiFi desligada
repouso	9227	95
andar	660	9340
correr	0	10000
andar de bicicleta	469	9531
andar de carro	191	9809

Tabela 6: Quantidade de amostras cara cada estado de WiFi por atividade

4.3 Experiências

De modo a obter os melhores resultados possíveis para cada um dos modelos, foi utilizado o *Grid search* ao treinar os modelos. Este é uma técnica utilizada para identificar hiperparâmetros ótimos para um modelo. Na prática o que esta técnica faz é criar modelos para cada combinação de hiperparâmetros e depois avaliar o melhor através de *cross-validation*. Este método de avaliação permite verificar o quão bem um modelo está a generalizar, já que consiste num método de reamostragem que usa diferentes partes dos dados para testar e treinar um modelo em diferentes iterações.

Para obter os hiperparâmetros ótimos, todos os modelos tiraram proveito da função *GridSearchCV* da biblioteca *scikit learn* de *python* com os seguintes parâmetros:

- ***cv:*** *KFold(n_splits=10)* - define a estratégia de divisão do *cross-validation*;
- ***n_jobs:*** *-1* - número de *jobs* a correr em paralelo. O valor -1 para cria um número máximo de *jobs* suportado pelo CPU;
- ***verbose:*** *2* - controla a verbosidade enquanto corre;
- ***return_train_score :*** *True* - boleano que permite retornar ou não o valor do treino;
- ***scoring:*** *accuracy* - estratégia para avaliar a performance do modelo criado pelo *cross-validation* no conjunto de teste;

- ***refit:*** *True* - reajusta um estimador usando os melhores parâmetros encontrados em todo o conjunto de dados.

4.3.1 Regressão logística

Para a criação do modelo de regressão logística foi utilizada a função *LogisticRegression* da biblioteca *scikit learn* de *python*. Para inicializar a função foram adicionados os seguintes parâmetros:

- ***solver:*** *saga* - permite regularização L1 e é a escolha normal para utilizar na regressão logística multi nominal esparsa;
- ***max_iter:*** 5000 - representa o número máximo de iterações para o *solver* convergir;
- ***random_state*** - permite controlar o gerador de números aleatórios e assim correndo exatamente o mesmo modelo para o mesmo conjunto de dados obtém-se o mesmo resultado.

Foram testados alguns hiperparâmetros para otimizar o modelo. Em seguida podem-se ver as opções utilizadas para estes:

- ***logisticregression__C****:* [0.009, 0.01, 0.09, 1, 5] - controla a força de regularização do modelo. Um valor mais pequeno resulta numa regularização mais forte, o que significa que o modelo será menos complexo e mais fácil de generalizar;
- ***logisticregression__penalty****:* [l1, l2] - adiciona um tipo de regularização ao modelo.

Utilizando a *grid search* obteve-se o valor 5 para *logisticregression__C* e a opção l1 para *logisticregression__penalty*.

4.3.2 Árvore de decisão

Foi utilizada a função *DecisionTreeClassifier* da biblioteca *scikit learn* de *python* para a criação do modelo de árvore de decisão. Para inicializar a função foi apenas utilizado o parâmetro *random_state* para ser possível obter sempre o mesmo resultado correndo o mesmo modelo para o mesmo conjunto de dados.

Foram testados alguns hiperparâmetros para otimizar o modelo. Em seguida podem-se ver as opções utilizadas para estes:

- ***criterion****: [gini, entropy]* - define a função que mede a qualidade das divisões;
- ***max_depth****: range(1,10)* - controla a profundidade máxima da árvore resultante;
- ***min_samples_leaf:*** *range(1,5)* - representa o número mínimo de amostras necessárias para ser um nodo folha.

Os hiperparâmetros ótimos obtidos pela *grid search* foram então o *entropy* para o *criterion*, uma *max_depth* de 9 e ainda um *min_samples_leaf* de 1.

4.3.3 *Random forest*

Também da biblioteca *scikit learn* de *python*, a função *RandomForestClassifier* foi aquela utilizada para a criação de um modelo de *random forest*. Nos seus parâmetros foi apenas passado um valor para o *random_state*, por motivos já anteriormente referidos.

Foram testados alguns hiperparâmetros para otimizar o modelo. Em seguida podem-se ver as opções utilizadas para estes:

- ***bootstrap****: True* - boleano que defeni se amostras de inicialização são usadas na construção das árvores. No caso de estar a falso, os dados todos são usados para construir cada árvore;
- ***max_depth****: [80, 90, 100, 110]* - controla a profundidade máxima da árvore resultante;
- ***max_features****: [2, 3]* - representa o número de *features* a serem consideradas quando se procura para a melhor divisão;
- ***min_samples_leaf:*** *[3, 4, 5]* - representa o número mínimo de amostras necessárias para ser um nodo folha;
- ***min_samples_split:*** *[8, 10, 12]* - define o número mínimo de amostras necessárias para dividir um nodo interno;
- ***n_estimators:*** *[100, 200, 300, 1000]* - representa o número de árvores da floresta.

Passados os hiperparâmetros ao *grid search*, obtiveram-se os seus valores ótimos de acordo com o que se pode observar na tabela 7.

hiperparâmetro	valor
bootstrap	*True*
max_depth	80
max_features	3
min_samples_leaf	3
min_samples_split	8
n_estimators	300

Tabela 7: Hiperparâmetros ótimos do modelo *random forest*

4.3.4 *XGBoost*

Para o modelo *xgboost* foi utilizada a função *XGBClassifier* da biblioteca *xgboost* de *python*. Nos seus parâmetros apenas foi adicionado um valor para o *random_state*.

O modelo foi corrido com uma diversidade de hiperparâmetros diferentes. Em seguida podemos ver quais desses foram alterados e os valores que foram testados:

- ***max_depth****: range(3, 10, 2)* - controla a profundidade máxima das árvores base;
- ***n_estimators:*** *[100, 250, 500, 1000]* - representa o número de árvores com gradiente aumentado; makes the boosting process more or less conservative
- ***learning_rate****: [0.001, 0.01, 0.1]* - torna o processo de *boosting*, mais ou menos conservativo;
- ***objective:*** *[binary:hinge, binary:logistic, binary:logitraw]* - especifica a tarefa de aprendizagem e o respetivo objetivo de aprendizagem.

Após a execução do *grid search* foram obtidos os valores ótimos presentes na tabela 8.

hiperparâmetro	valor
max_depth	9
n_estimators	100
learning_rate	0.1
objective	*binary:logistic*

Tabela 8: Hiperparâmetros ótimos do modelo *xgboost*

4.4 Sumário

Após toda a preparação de dados obteve-se o conjunto de *features* que se pode observar na tabela 9. Estas foram então as *features* que foram usadas para treinar os modelos, à exceção da *feature activity* que representa a variável dependente do nosso problema. As variáveis independentes foram por sua vez divididas em dois conjuntos, o conjunto de treino e o conjunto de testes. O conjunto de treino é aquele que serve para treinar o modelo, enquanto que a amostra de teste é aquela utilizada para o avaliar e obter no fim as métricas de precisão. A divisão entre treino e teste é realizada de forma aleatória ficando no fim 80% dos dados para treino e os restantes para teste.

Feature	Descrição
accelerometerx_diff_rolling_mean	Média acelerómetro no eixo do x
accelerometery_diff_rolling_mean	Média acelerómetro no eixo do y
accelerometerz_diff_rolling_mean	Média acelerómetro no eixo do z
gravityx_diff_rolling_mean	Média sensor de gravidade no eixo do x
gravityy_diff_rolling_mean	Média sensor de gravidade no eixo do y
gravityz_diff_rolling_mean	Média sensor de gravidade no eixo do z
gyroscopex_diff_rolling_mean	Média giroscópio no eixo do x
gyroscopey_diff_rolling_mean	Média giroscópio no eixo do y
gyroscopez_diff_rolling_mean	Média giroscópio no eixo do z
orientationx_diff_rolling_mean	Média sensor de orientação no eixo do x
orientationy_diff_rolling_mean	Média sensor de orientação no eixo do y
linear_accelerometer_magnitude_rolling_mean	Média magnitude da aceleração linear
linear_gravity_magnitude_rolling_mean	Média magnitude da gravidade linear
accelerometerz_diff_rolling_min	Mínimo acelerómetro no eixo do z
gravityx_diff_rolling_min	Mínimo sensor de graviade no eixo do x
gravityy_diff_rolling_min	Mínimo sensor de graviade no eixo do y
gravityz_diff_rolling_min	Mínimo sensor de graviade no eixo do z
gyroscopex_diff_rolling_min	Mínimo giroscópio no eixo do x
gyroscopey_diff_rolling_min	Mínimo giroscópio no eixo do y
gyroscopez_diff_rolling_min	Mínimo giroscópio no eixo do z
orientationx_diff_rolling_min	Mínimo sensor de orientação no eixo do x
orientationy_diff_rolling_min	Mínimo sensor de orientação no eixo do y
linear_gravity_magnitude_rolling_min	Mínimo magnitude da gravidade linear
accelerometerz_diff_rolling_max	Máximo acelerómetro no eixo do z
gravityx_diff_rolling_max	Máximo sensor de gravidade no eixo do x
gravityy_diff_rolling_max	Máximo sensor de gravidade no eixo do y
gravityz_diff_rolling_max	Máximo sensor de gravidade no eixo do z
orientationx_diff_rolling_max	Máximo sensor de orientação no eixo do x
orientationy_diff_rolling_max	Máximo sensor de orientação no eixo do y
activity	Atividade representada na amostra

Tabela 9: *Features* do modelo

Capítulo 5

Resultados e discussão

No presente capítulo serão discutidos os resultados obtidos. Numa primeira fase será realizada uma análise à aplicação de recolha de dados. Depois será realizada uma análise individual a cada um dos modelos desenvolvidos, assim como uma discussão face à sua capacidade de prever cada classe individualmente. Para fechar o capítulo será realizada uma comparação de mais alto nível entre cada um dos modelos desenvolvidos.

5.1 Aplicação de recolha de dados

A aplicação de recolha de dados desenvolvida revelou-se essencial ao projeto, sem a qual não seria possível realizar nenhuma das etapas seguintes. Esta encontrou-se sempre *online* ao longo do desenvolvimento do projeto, facilitando assim a recolha de dados, uma vez que qualquer pessoa na sua posse, poderia recolher dados sem qualquer dificuldade nem restrição. Este fator revelou-se fundamental para que fosse possível a recolha de uma quantidade significativa de dados.

Um problema que surgiu foi a fraca adesão de pessoas à utilização da aplicação, criando assim uma comunidade de recolha de dados muito reduzida o que levou a uma menor variabilidade entre dados recolhidos para cada tipo de atividade. Este facto foi notório ao analisar os padrões de utilização de *bluetooth* e tipo de conectividade para cada uma das atividades, que se revelavam praticamente sempre os mesmos dentro de cada atividade.

5.2 Modelos *Machine Learning*

5.2.1 Regressão logística

A regressão logística é um dos algoritmos mais simples, sendo normalmente mais indicado para problemas mais simples. Assim é expectável que outros algoritmos obtenham melhores resultados em problemas de uma maior complexidade, como é o caso. Mesmo assim o modelo criado com recurso a este algoritmo obteve uma *accuracy* de 0.845, que já é um valor respeitável. Observando a tabela 10 e a figura 33 é possível analisar a *performance* do modelo para cada uma das atividades. Desta forma podemos ver que este modelo esteve bastante bem a identificar a atividade de correr, isto é, tem uma *precision* bastante alta, no valor de 0.93. No entanto o modelo teve algumas dificuldades a identificar outras atividades, destacando-se a atividade de andar com apenas 0.77 de *precision*.

classe	*precision*	*recall*	*f1-score*	nº amostras
repouso	0.84	0.93	0.88	1849
andar	0.77	0.67	0.72	1987
correr	0.93	0.90	0.92	1988
andar de bicicleta	0.82	0.80	0.81	2024
andar de carro	0.86	0.92	0.89	2017

Tabela 10: Tabela de resultados Regressão logística

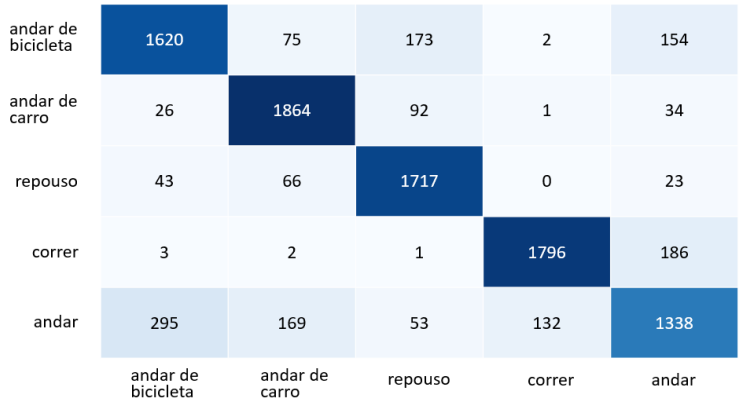


Figura 33: Matriz de confusão da regressão logística

Podemos ainda observar quais foram as 10 *features* mais relevantes para este modelo na tabela 11. Desta forma é possível verificar que a *feature* mais impactante foi a *linear_accelerometer_magnitude_rolling_mean*, seguida já com alguma distância, pela *linear_gravity_magnitude_rolling_mean* que apresenta apenas metade da importância da primeira. As *features* seguintes apresentam já uma maior equivalência em termos de importância, sendo a diferença entre a terceira e a décima de apenas 0.049.

feature	importância
linear_accelerometer_magnitude_rolling_mean	0.400
linear_gravity_magnitude_rolling_mean	0.202
gyroscopez_diff_rolling_min	0.095
gyroscopex_diff_rolling_min	0.093
gravityz_diff_rolling_max	0.086
accelerometerz_diff_rolling_min	0.074
linear_gravity_magnitude_rolling_min	0.059
accelerometerz_diff_rolling_max	0.054
gravityy_diff_rolling_max	0.048
gyroscopey_diff_rolling_min	0.046

Tabela 11: Feature importance regressão logística

5.2.2 Árvore de decisão

As árvores de decisão são algoritmos que suportam uma decisão hierarquia com uma estrutura em árvore. Este modelo obteve uma *accuracy* de 0.893, revelando-se superior ao modelo da regressão linear. Através da tabela 12 e da figura 34 conseguimos observar que a classe de repouso é aquela que o modelo consegue prever com maior facilidade, com 0.97 de *precision*. Semelhante ao modelo de regressão linear, a classe mais complicada de prever é andar, com um modesto valor de 0.77 para esta métrica.

classe	*precision*	*recall*	*f1-score*	nº amostras
repouso	0.97	0.93	0.95	1849
andar	0.77	0.84	0.80	1987
correr	0.96	0.93	0.94	1988
andar de bicicleta	0.85	0.85	0.85	2024
andar de carro	0.95	0.92	0.93	2017

Tabela 12: Tabela de resultados Árvore de decisão

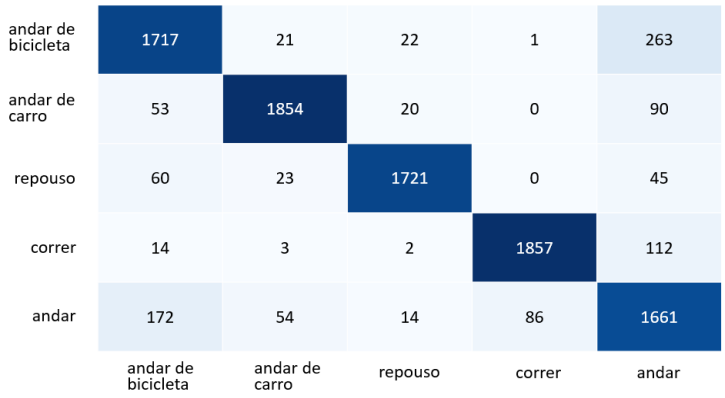


Figura 34: Matriz de confusão da árvore de decisão

Relativamente à importância das *features*, a *linear_accelerometer_magnitude_rolling_mean* revelou-se tal como no modelo anterior, a *feature* com maior importância com um valor de 0.278. Esta no entanto é seguida de perto pela *feature gyroscopex_diff_rolling_min* com um valor de 0.230. O *top* três é fechado com um valor de 0.116 para a *feature gravityy_diff_rolling_max*. As restantes *features*, com menor importância, não revelam grandes diferenças entre si.

feature	importância
linear_accelerometer_magnitude_rolling_mean	0.278
gyroscopex_diff_rolling_min	0.230
gravityy_diff_rolling_max	0.116
gravityy_diff_rolling_min	0.067
gravityz_diff_rolling_max	0.049
accelerometerz_diff_rolling_max	0.047
gyroscopez_diff_rolling_min	0.046
linear_gravity_magnitude_rolling_mean	0.036
gravityz_diff_rolling_min	0.027
accelerometerz_diff_rolling_min	0.018

Tabela 13: Feature importance àrvore de decisão

5.2.3 *Random forest*

O *random forest* é um algoritmo semelhante às árvores de decisão, com a diferença de em vez de uma árvore deste tipo, gerar um conjunto delas. A *accuracy* obtida por este modelo foi de 0.955, melhor que o resultado do modelo da árvore de decisão, algo que era expectável, visto este algoritmo representar uma evolução face ao anterior. A partir da tabela 14 e da figura 35, verifica-se que classe com melhor *precision*, com um valor de 0.99, foi a atividade de repouso. Por sua vez a classe mais complicada de prever, semelhante a ambos os modelos anteriores, foi a atividade de andar, com um valor de 0.90 para a mesma métrica.

classe	*precision*	*recall*	*f1-score*	nº amostras
repouso	0.99	0.97	0.98	1849
andar	0.90	0.94	0.92	1987
correr	0.98	0.96	0.97	1988
andar de bicicleta	0.95	0.95	0.95	2024
andar de carro	0.97	0.96	0.96	2017

Tabela 14: Tabela de resultados *random forest*

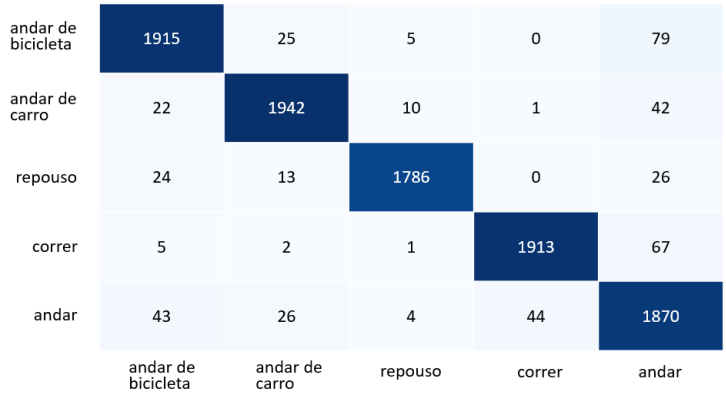


Figura 35: Matriz de confusão da *random forest*

Neste modelo a *feature linear_accelerometer_magnitude_rolling_mean* com um valor de 0.112 volta a ser aquela com uma maior importância. No entanto, em contraste com os modelos anteriores podemos ver que todas as outras *features* do *top* 10 apresentam valores não muito distantes entre si, mas bastante inferiores à *feature* mais importante.

feature	importância
linear_accelerometer_magnitude_rolling_mean	0.112
gravityy_diff_rolling_max	0.022
accelerometerz_diff_rolling_max	0.022
gravityy_diff_rolling_min	0.021
accelerometerz_diff_rolling_min	0.020
gyroscopex_diff_rolling_min	0.020
gyroscopez_diff_rolling_min	0.017
gyroscopey_diff_rolling_min	0.011
gravityz_diff_rolling_max	0.009
gravityz_diff_rolling_min	0.009

Tabela 15: *Feature importance random forest*

5.2.4 *XGboost*

Para finalizar, criou-se um modelo utilizando o algoritmo *XGboost*. Este algoritmo representa uma evolução do algoritmo de *random forest*, na medida em que enquanto o segundo apresenta um conjunto fixo de parâmetros, os parâmetros do *XGboost* vão-se ajustando iterativamente ao longo do treino. Desta forma esperava-se uma melhoria nos resultados, o que de facto se verificou, já que este obteve uma *accuracy* de 0.979. Através da tabela 16 e da figura 36 averigua-se que este modelo obteve resultados bastante impressionantes, sendo a classe com menor *precision* a atividade de andar com um valor de 0.95.

classe	*precision*	*recall*	*f1-score*	nº amostras
repouso	0.99	0.99	0.99	1849
andar	0.95	0.97	0.96	1987
correr	0.99	0.98	0.99	1988
andar de bicicleta	0.97	0.98	0.98	2024
andar de carro	0.99	0.98	0.98	2017

Tabela 16: Tabela de resultados do xgboost

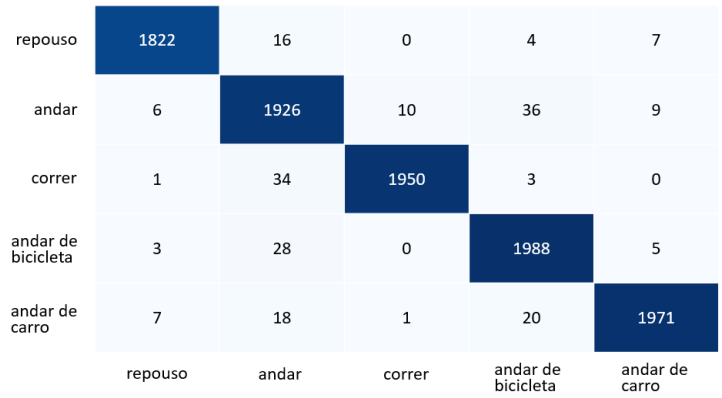


Figura 36: Matriz de confusão do *xgboost*

Tal como o modelo da *random forest* a *feature linear_accelerometer_magnitude_rolling_mean* lidera em termos de importância, com um valor de 0.226 enquanto que todas as outras obtiveram uma importância significativamente inferior.

feature	importância
linear_accelerometer_magnitude_rolling_mean	0.226
gravityy_diff_rolling_min	0.031
gyroscopez_diff_rolling_min	0.027
gyroscopex_diff_rolling_min	0.021
gravityz_diff_rolling_max	0.014
gravityy_diff_rolling_max	0.014
accelerometerz_diff_rolling_max	0.012
gravityz_diff_rolling_min	0.008
accelerometerz_diff_rolling_min	0.004
gyroscopey_diff_rolling_min	0.004

Tabela 17: Feature importance XGBoost

5.3 Comparação

Observando agora na tabela 18 a *accuracy* final de cada um dos modelos, podemos concluir que o melhor modelo foi o *XGboost* com um valor de 0.979. O último modelo revelou-se então o melhor, uma vez que neste projeto houve uma tentativa de modelo após modelo, tentar encontrar algoritmos cada vez mais capazes, até chegar a um resultado final que fosse considerado bastante bom.

modelo	*accuracy*
regressão logística	0.845
árvores de decisão	0.893
random forests	0.955
xgboost	0.979

Tabela 18: Comparação da *accuracy* dos modelos

Capítulo 6

Conclusão

Este capítulo terá o seu foco na conclusão do projeto, bem como nas perspectivas para trabalhos futuros, além de abordar alguns pensamentos finais.

6.1 Conclusões

Ao longo da minha jornada de pesquisa sobre "Reconhecimento de atividades em *smartphones* usando sensores não intrusivos", embarquei numa busca prática para melhorar a nossa compreensão das atividades humanas no contexto da utilização de *smartphones*. Esta tese girou em torno do desenvolvimento de uma estrutura abrangente que abrange coleção de dados, armazenamento de dados baseado em *cloud*, pré-processamento de dados e ainda desenvolvimento de modelo de *machine learning*.

Na minha busca para colecionar dados de *smartphones* de forma transparente, criei uma aplicação *android*, que não serviu apenas como uma ferramenta robusta de aquisição de dados, mas também garantiu a continuidade da mesma. Simultaneamente, criei uma aplicação em *cloud* na plataforma *Google Cloud Platform*, fornecendo um repositório seguro e acessível para os dados dos sensores colecionados. Esta sinergia entre a recolha de dados e o armazenamento na nuvem não só simplificou o processo de investigação, mas também lançou as bases para as fases subsequentes do meu trabalho.

A transformação de dados brutos de sensores em dados processados relevantes ao problema, foi sem dúvida um marco fundamental nesta jornada. O rigoroso pré-processamento de dados e técnicas de *feature engineering* foram aplicados meticulosamente, permitindo-me preencher a lacuna entre os dados brutos dos sensores e o desenvolvimento de modelos com capacidade de previsão. Esta etapa foi fundamental para dar sentido aos dados, garantindo a sua interpretabilidade e facilitando a análise subsequente.

No final desta pesquisa, o foco mudou para modelos de *machine learning*. Aproveitando os dados pré-processados, criei modelos complexos com a capacidade de distinguir entre diversas atividades, incluindo descansar, andar, correr, andar de bicicleta e andar de carro, com notável precisão. Estes modelos não só sublinham a eficácia da minha abordagem, mas também mostram o potencial para aplicações práticas em vários domínios, desde os cuidados de saúde, transportes, entre outros. Além disso, gostaria de referir que submeti um artigo com o título "*Activity recognition in smartphones using non-intrusive sensors*"na conferência ICAART, o qual resume toda a pesquisa realizada.

6.2 Trabalho futuro

Embora esta tese represente um avanço no campo do reconhecimento de atividades, é importante reconhecer que há mais trabalho a ser feito. Existem muitas possibilidades intrigantes para pesquisas futuras. Pretendo melhorar os modelos existentes, explorar novas atividades e expandir esta abordagem para uma gama mais ampla de dispositivos equipados com sensores, aumentando a sua flexibilidade e aplicabilidade em diversas situações. Para além disso penso ser importante numa fase inicial de recolha de dados, tentar recolher dados mais diversificados para cada tipo de atividade, de modo a possibilitar uma melhor generalização do modelo final.

6.3 Pensamentos finais

Em resumo, esta tese representa uma contribuição substancial para o domínio do reconhecimento de atividades em *smartphones*. Este trabalho ilustra a eficácia de uma abordagem abrangente que integra coleção de dados, armazenamento em *cloud*, pré-processamento de dados e *machine learning*. Com a evolução contínua da tecnologia dos *smartphones* e a importância crescente da interação humano-máquina, o caminho a seguir promete desafios e oportunidades à medida que nos esforçamos para tornar os nossos companheiros digitais mais inteligentes e mais sintonizados com a nossa vida quotidiana.

Bibliografia

[1] E. Tucker e Z. Miller. "How many people have smartphones in 2022?" Em: *Oberlo* (2022). url: https://www.oberlo.com/statistics/how-many-people-have-smartphones.

[2] S. Reinberg. "More Evidence Fitness Trackers Can Boost Your Health". Em: *WebMD* (2022). url: https://www.webmd.com/fitness-exercise/news/20220726/more-evidence-fitness-trackers-can-boost-your-health#1.

[3] D. Nield. "All the Sensors in Your Smartphone, and How They Work". Em: *Gizmodo* (2020). url: https://gizmodo.com/all-the-sensors-in-your-smartphone-and-how-they-work-1797121002.

[4] A. Vaughn, P. Biocco, Y. Liu e M. Anwar. "Activity Detection and Analysis Using Smartphone Sensors". Em: *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. 2018, pp. 102–107. doi: 10.1109/IRI.2018.00022.

[5] B. Fernandes e C. Analide. *Concepts and Platforms*. 2022.

[6] *Smart cities*. url: https://commission.europa.eu/eu-regional-and-urban-development/topics/cities-and-urban-development/city-initiatives/smart-cities_en#:~:text=Related%20links-,What%20are%20smart%20cities%3F,resource%20use%20and%20less%20emissions. (acedido em 09/12/2022).

[7] *Physical sensors*. url: https://sites.unica.it/dealab/physical-sensors/#:~:text=A%20physical%20sensor%20is%20a,observer%20or%20by%20an%20instrument. (acedido em 15/11/2022).

[8] D. Martin, N. Kühl e G. Satzger. "Virtual sensors". Em: *Business & Information Systems Engineering* 63.3 (2021), pp. 315–323.

[9] T. Brezmes, J.-L. Gorricho e J. Cotrina. "Activity recognition from accelerometer data on a mobile phone". Em: *International work-conference on artificial neural networks*. Springer. 2009, pp. 796–799.

[10] S. Sharma. *What Is Accelerometer? How to Use Accelerometer in Mobile Devices?* 2020. url: https://www.credencys.com/blog/accelerometer/ (acedido em 16/10/2022).

[11] N. Ravi, N. Dandekar, P. Mysore e M. L. Littman. “Activity recognition from accelerometer data”. Em: *Aaai*. Vol. 5. 2005. Pittsburgh, PA. 2005, pp. 1541–1546.

[12] R. Goodrich. *Accelerometers: What They Are How They Work*. 2013. url: https://www.livescience.com/40102-accelerometers.html#:~:text=The%20capacitance%20accelerometer%20senses%20changes,capacitance%20to%20voltage%20for%20interpretation. (acedido em 16/10/2022).

[13] J.-W. Joo e S.-H. Choa. “Deformation Behavior of MEMS Gyroscope Sensor Package Subjected to Temperature Change”. Em: *IEEE Transactions on Components and Packaging Technologies* 30.2 (2007), pp. 346–354. doi: 10.1109/TCAPT.2007.897948.

[14] B. Coley, B. Najafi, A. Paraschiv-Ionescu e K. Aminian. “Stair climbing detection during daily physical activity using a miniature gyroscope”. Em: *Gait & posture* 22.4 (2005), pp. 287–294.

[15] A. P. J. Dwiyantoro, I. G. D. Nugraha e D. Choi. “A simple hierarchical activity recognition system using a gravity sensor and accelerometer on a smartphone”. Em: *International Journal of Technology* 7.5 (2016), pp. 831–839.

[16] *Google Android Sensors*. https://source.android.com/docs/core/interaction/sensors/sensor-types. Accessed: 2022-10-18.

[17] *Tizen Docs Device Sensors*. https://docs.tizen.org/application/native/guides/location-sensors/device-sensors/. Accessed: 2022-10-18.

[18] S. Haldar. “Chapter 5 - Exploration Geophysics”. Em: *Mineral Exploration*. Ed. por S. Haldar. Boston: Elsevier, 2013, pp. 73–93. isbn: 978-0-12-416005-7. doi: https://doi.org/10.1016/B978-0-12-416005-7.00005-2. url: https://www.sciencedirect.com/science/article/pii/B9780124160057000052.

[19] C.-C. Lin, C.-C. Chang, D. Liang e C.-H. Yang. “A New Non-Intrusive Authentication Method Based on the Orientation Sensor for Smartphone Users”. Em: *2012 IEEE Sixth International Conference on Software Security and Reliability*. 2012, pp. 245–252. doi: 10.1109/SERE.2012.37.

[20] S. Hughes, T. D’Arcy, D. Maxwell, W. Chiu, A. Milner, J. Saunders e R. Sheppard. “Volume estimation from multiplanar 2D ultrasound images using a remote electromagnetic position and orientation sensor”. Em: *Ultrasound in Medicine Biology* 22.5 (1996), pp. 561–572. issn: 0301-5629. doi: https://doi.org/10.1016/0301-5629(96)00022-1. url: https://www.sciencedirect.com/science/article/pii/0301562996000221.

[21] Y. Zhao, L. Görne, I.-M. Yuen, D. Cao, M. Sullman, D. Auger, C. Lv, H. Wang, R. Matthias, L. Skrypchuk e A. Mouzakitis. “An Orientation Sensor-Based Head Tracking System for Driver Behaviour Monitoring”. Em: *Sensors* 17.11 (2017). issn: 1424-8220. doi: 10.3390/s17112692. url: https://www.mdpi.com/1424-8220/17/11/2692.

[22] S. Dutta. "Point of care sensing and biosensing using ambient light sensor of smartphone: Critical review". Em: *TrAC Trends in Analytical Chemistry* 110 (2019), pp. 393–400. issn: 0165-9936. doi: https://doi.org/10.1016/j.trac.2018.11.014. url: https://www.sciencedirect.com/science/article/pii/S0165993618302978.

[23] C Rüth, A Vogler e W Karsten. "Ambient Light Sensors General Application Note". Em: *OSRAM Opto Semiconductors GmbH, Regensburg, Germany* (2006).

[24] M. Ramteke. *Ambient light sensor - working amp; its applications*. 2022. url: https://www.semiconductorforu.com/ambient-light-sensor-working-its-applications/ (acedido em 31/10/2022).

[25] D. R. Hidayat e W. Windarjoto. "Comparison of proximity sensor responsiveness levels on Samsung smartphones". Em: *Linguistics and Culture Review* 6 (2022), pp. 194–200.

[26] Electricalvoice. *7 proximity sensor applications - you should know*. 2021. url: https://electricalvoice.com/proximity-sensor-applications/ (acedido em 07/11/2022).

[27] Ramzy. *5 types of proximity sensors (application and advantages)*. 2022. url: https://www.linquip.com/blog/types-of-proximity-sensors/#Common_Applications-2 (acedido em 07/11/2022).

[28] M. Budimir. *What are photoelectric proximity sensors?* url: https://www.motioncontroltips.com/what-are-photoelectric-proximity-sensors/#:~:text=One%20of%20the%20most%20common,back%20from%20an%20object%27s%20surface. (acedido em 07/11/2022).

[29] *What is bluetooth and how do I use it?* 2020. url: https://www.samsung.com/uk/support/mobile-devices/what-is-bluetooth/ (acedido em 07/11/2022).

[30] Z. Chen, Y. Chen, L. Hu, S. Wang, X. Jiang, X. Ma, N. D. Lane e A. T. Campbell. "ContextSense: unobtrusive discovery of incremental social context using dynamic bluetooth data". Em: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. 2014, pp. 23–26.

[31] G. M. Harari, N. D. Lane, R. Wang, B. S. Crosier, A. T. Campbell e S. D. Gosling. "Using Smartphones to Collect Behavioral Data in Psychological Science: Opportunities, Practical Considerations, and Challenges". Em: *Perspectives on Psychological Science* 11.6 (2016). PMID: 27899727, pp. 838–854. doi: 10.1177/1745691616650285. eprint: https://doi.org/10.1177/1745691616650285. url: https://doi.org/10.1177/1745691616650285.

[32] J. P. Mueller e L. Massaron. *Machine learning for dummies*. John Wiley & Sons, 2021.

[33] P. Novais. *Dados e Aprendizagem Automática - Machine Learning*. 2021.

[34] B. Fernandes, F. Gonçalves, V. Alves e C. Analide. *Dados e Aprendizagem Automática - Métricas de Qualidade e Validação de Modelos*. 2021.

[35] P. Novais. *Dados e Aprendizagem Automática - Supervised Learning Linear Logistic Regression*. 2021.

[36] P. Novais. *Dados e Aprendizagem Automática - Support Vector Machine*. 2021.

[37] V. Alves e F. Ferraz. *Aprendizagem Profunda Artificial Neural Network*. 2022.

[38] P. Novais. *Dados e Aprendizagem Automática - Segmentação*. 2021.

[39] X. Su, H. Tong e P. Ji. "Activity recognition with smartphone sensors". Em: *Tsinghua Science and Technology* 19.3 (2014), pp. 235–249. doi: 10.1109/TST.2014.6838194.

[40] A. Bayat, M. Pomplun e D. A. Tran. "A study on human activity recognition using accelerometer data from smartphones". Em: *Procedia Computer Science* 34 (2014), pp. 450–457.

[41] H. Chen, S. H. Cha e T. W. Kim. "A framework for group activity detection and recognition using smartphone sensors and beacons". Em: *Building and Environment* 158 (2019), pp. 205–216.

[42] X. Wang e H. Kim. "Detecting user activities with the accelerometer on android smartphones". Em: *Journal of Multimedia Information System* 2.2 (2015), pp. 233–240.

[43] A. Krivov e A. Topalidis. *Spring boot vs quarkus: Which framework to choose in 2022*. 2022. url: https://maddevs.io/blog/spring-boot-vs-quarkus/ (acedido em 29/12/2022).

Anexo I

Publicação

I.1 Activity recognition in smartphones using non-intrusive sensors

Authors: Pedro Fernandes, Bruno Fernandes, Cesar Analide

Title: Activity recognition in smartphones using non-intrusive sensors

Conference: International Conference on Agents and Artificial Intelligence (ICAART) 2024

Date submission: 2023, October

Abstract: Activity recognition using smartphones has gained increased attention in recent years due to the widespread adoption of these devices and, consequently, their various sensors. These sensors are capable of providing very relevant data for this purpose. Non-intrusive sensors, in particular, offer the advantage of collecting data without requiring the user to perform any specific action or use any additional devices. The objective of this study was, therefore, the development of an application designed for activity recognition using exclusively non-intrusive sensors available in any smartphone. The data collected by these sensors under- went several processing stages, and after numerous iterations, a set of highly favorable features for training the machine learning models was obtained. The most prominent result was achieved by the model using the XGBoost algorithm, which achieved an impressive accuracy rate of 0.979. This quite robust result confirms the high effectiveness of using this type of sensors for activity recognition.

Keywords: Activity recognition, Human behavior, Machine learning, Mobile device, Sensor data and Smartphones