INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Sistema automático de melhoria contínua para deteção e identificação de sons impulsivos

André da Silva Mendes

Licenciado em Engenharia Informática e de Computadores

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadores : Professor Doutor Paulo Trigo
Professor Doutor Joel P. Paulo

Júri:

Presidente: Professor Doutor Tiago Miguel Braga da Silva Dias

Vogais: Professor Doutor Gonçalo Caetano Marques
Professor Doutor Paulo Trigo Cândido da Silva

Fevereiro, 2024

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Sistema automático de melhoria contínua para deteção e identificação de sons impulsivos

André da Silva Mendes

Licenciado em Engenharia Informática e de Computadores

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadores : Professor Doutor Paulo Trigo
Professor Doutor Joel P. Paulo

Júri:

Presidente: Professor Doutor Tiago Miguel Braga da Silva Dias

Vogais: Professor Doutor Gonçalo Caetano Marques
Professor Doutor Paulo Trigo Cândido da Silva

Fevereiro, 2024

Agradecimentos

Aos meus orientadores, Prof. Doutor Paulo Trigo, e Prof. Doutor Joel Paulo, sem os quais a realização deste trabalho não seria possível.

Ao Laboratório de Áudio e Acústica do ISEL, pela disponibilização do espaço e equipamentos.

À minha familia, em particular, à minha companheira Tatiana Santos, por todo o apoio durante a realização deste trabalho.

Resumo

A deteção e identificação de sons impulsivos aplicados num contexto específico, nomeadamente em eventos desportivos, possibilitam a análise e síntese de várias métricas e estatísticas associadas ao jogo, ou mesmo, ao desempenho do jogador. Neste contexto, a identificação automática de um som impulsivo, como o som da batida da bola na raquete do jogador, é uma contribuição importante para a criação de uma fonte de dados na qual podem ser realizadas análises específicas do jogo.

Considerar todas as características de um determinado tipo de som impulsivo em várias condições e ambientes, envolve lidar com bastantes variáveis, sendo igualmente difícil encontrar, de forma eficiente, os valores dos híper-parâmetros que permitem obter a melhor configuração para um determinado algoritmo a ser executado num processo de aprendizagem automática (*machine learning*).

O contributo deste trabalho é o de explorar o espaço de híper-parâmetros (HyP) na procura dos valores que otimizem o desempenho de todo o processo de classificação automática de sons impulsivos. Esse processo inicia na geração do conjunto de dados (*dataset*) a processar, prossegue com a aprendizagem de modelos de classificação, e termina com a avaliação dos modelos aprendidos. Neste trabalho propõe-se estender o conceito de HyP de modo a englobar também a construção do próprio *dataset*.

Nas experiências realizadas, é considerado um problema de classificação binária, onde é necessário fazer a distinção entre o evento pretendido e o ruído. A validação do processo recorre a áudio extraído de vídeos de competições de eventos desportivos, nomeadamente, ténis e padel, onde se procuram identificar os sons de pancadas da raquete na bola. Os resultados experimentais, onde se observa uma taxa de sucesso (*accuracy*) do modelo a rondar os 93%, permitem evidenciar a influência das características do som na construção do *dataset* e no desempenho global do processo de classificação automática de sons impulsivos.

Palavras-chave: Inteligência Artificial; Aprendizagem Automática; Análise de áudio; *Dataset* de Eventos Sonoros; Otimização de Híper-parâmetros.

Abstract

The detection and identification of impulsive sounds applied in a specific context, particularly in sports events, enables the analysis and synthesis of various metrics and statistics associated with the game or even the player's performance. In this context, the automatic identification of an impulsive sound, such as a ball being hit by a player, is a major contribution to the construction of a data source on which game-specific analysis can be performed.

Considering all the characteristics (features) of a particular type of impulsive sound in various conditions/environments involves dealing with numerous variables, making it equally challenging to efficiently find the values of the hyperparameters that allow obtaining the best configuration for a given algorithm to be executed in a machine learning process.

The contribution of this work is to explore the hyperparameter (HyP) space in search of values that optimize the performance of the entire process of automatic impulsive sound classification. This process begins with the generation of the dataset to be processed, continues with the training of classification models, and ends with the evaluation of the learned models. In this work, we propose to extend the concept of HyP to also encompass the construction of the dataset itself.

The experiments consider a binary classification problem, where a distinction must be made between the intended event and noise. The validation of the process resorts to an audio extracted from videos of sports event competitions, specifically in tennis and padel, where the goal is to identify the sounds of racket hits on the ball. The experiment results, where the model's "accuracy" is observed to be around 93%, enable to evidence the impact of the influence of sound characteristics on the construction of the dataset and on the overall performance of the process of automatic classification of impulsive sounds.

Keywords: Artificial Intelligence; Machine Learning; Audio Analysis; Sound Event Dataset; Hyperparameter Optimization.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Listagens

1

Introdução

1.1 Enquadramento

Em geral na prática de modalidades desportivas, o desempenho de um atleta pode melhorar quando ele tem uma perspetiva externa da sua atividade. Essa perspetiva pode ser dada pelo seu treinador. Complementarmente, ou mesmo quando a supervisão humana não é possível, essa análise só pode ser feita vendo ou ouvindo novamente a gravação da atividade. Como tal, é de grande utilidade uma ferramenta para a deteção e identificação de sons impulsivos aplicada, em particular, a eventos desportivos (por exemplo, na modalidade de ténis e padel, onde se procura identificar os sons de pancadas da raquete na bola). Neste trabalho, propõe-se o desenvolvimento de um sistema para identificação automática de sons impulsivos aplicado a eventos desportivos. O objetivo é criar uma fonte de dados com o instante de tempo de cada evento (pancada da raquete na bola). Isso representa os dados-base para análises automáticas específicas do jogo.

Um som impulsivo pode ser caracterizado pela sua curta duração, onde a sua energia tem um crescimento rápido inicialmente (*onset or attack*) seguido de um decaimento (*decay / release*) que depende do ambiente acústico onde o som é gerado. A reverberação do espaço tem uma grande influência na duração do som impulsivo, no entanto, existem outras características que é necessário extrair do sinal acústico de forma a distinguir e identificar um som impulsivo. Por exemplo, as características associadas à energia do sinal num intervalo. Todas essas características têm impacto na geração do

conjunto de dados (*dataset*) a processar. Essas variáveis, assim como as variáveis associadas à escolha do algoritmo de classificação a utilizar, representam um vasto conjunto de híper-parâmetros (HyP), que geram um espaço de pesquisa de grande dimensão.

Neste trabalho propõe-se estender o conceito de HyP de modo a englobar também a construção do próprio *dataset*.

Por maior que seja o conhecimento e experiência na área do áudio e da aprendizagem automática, é praticamente impossível prever, à partida, a configuração dos HyP mais adequada ao problema. Além disso, os algoritmos de aprendizagem profunda (*deep-learning*) têm demonstrado bons resultados, contudo, esses algoritmos têm um espaço de pesquisa (*search space*) ainda maior do que os algoritmos tradicionais de aprendizagem automática (*machine learning*). Nesse sentido, este trabalho está focado na exploração do espaço de HyP na procura dos valores que otimizem o desempenho de todo o processo de classificação automática de sons impulsivos. Esse processo inicia na geração do conjunto de dados (*dataset*) a processar, prossegue com a aprendizagem de modelos de classificação, e termina com a avaliação dos modelos aprendidos.

Coloca-se como hipótese de trabalho, a melhoria do desempenho global do processo, resultante da extensão do conceito de HyP à construção do *dataset*.

1.2 Principais contributos

Este trabalho tem os seguintes contributos:

- Sistema para a exploração do espaço de híper-parâmetros na otimização de todo o processo de classificação automática de sons impulsivos, estendendo o conceito de híper-parâmetro de modo a englobar também a construção do próprio *dataset*;
- Projeto de implementação do sistema, disponibilizado à comunidade;
- Os contributos e suporte experimental da hipótese de trabalho, deram origem às seguintes 3 publicações:
  - André Mendes, Paulo Trigo, e Joel Paulo, "Hyperparameter Optimization for Impulsive Sound Classifiers", *International Conference on Industry Sciences and Computer Science Innovation, ELSEVIER B.V.*, 2023.
  - André Mendes, Paulo Trigo, e Joel Paulo, "Otimização de Híper-parâmetros em Classificadores de Som", *TecniAcústica*, 2023.

  - André Mendes, Paulo Trigo, e Joel Paulo, "Hyperparameter Optimization for the Entire Classification Process of Impulsive Sounds", *International Conference on Agents and Artificial Intelligence*, 2024.

- A proposta de trabalho futuro foi concretizada com a apresentação de uma ideia de projeto, na Unidade Curricular de Projeto 2023/2024 da Licenciatura em Engenharia Informática e Multimédia, no Instituto Superior de Engenharia de Lisboa (ISEL): "Ambiente interativo para configuração de híper-parâmetros em classificadores - o caso dos sons impulsivos".

1.3 Organização do documento

O restante documento está organizado da seguinte forma:

- No capítulo 2 é apresentado o estado da arte; algumas informações são sustentadas com trabalhos científicos na área da exploração de híper-parâmetros e da classificação de sons impulsivos.
- O capítulo 3 é dividido em duas partes principais. Na primeira parte do capítulo são apresentados os fundamentos que sustentam o trabalho realizado: o som e as suas características, os algoritmos para classificação binária, e as técnicas de exploração do espaço de híper-parâmetros. Na segunda parte do capítulo é apresentada a metodologia de todo o processo de classificação automática de sons impulsivos, desde a recolha dos dados até à geração e avaliação dos modelos.
- No capítulo 4 são apresentados os detalhes da implementação do sistema; identificadas as opções teóricas e justificadas as dependências tecnológicas.
- No capítulo 5 são apresentados os resultados experimentais que suportam a hipótese de trabalho.
- No capítulo 6 são apresentadas as principais conclusões, e deixadas algumas sugestões de trabalho futuro, visando a continuidade deste trabalho.
- Por fim, no apêndice A é referenciado o repositório onde está o projeto de implementação do sistema.

2

Estado da arte

A utilização de abordagens baseadas na aprendizagem automática no tema da deteção e identificação de sinais acústicos, mais concretamente, na classificação de sons impulsivos, tem tido bastante investigação e avanços significativos [3, 4, 15, 23, 24, 26, 28, 30]. Têm sido propostas varias técnicas para a detecção e identificação de sons impulsivos. No trabalho [4] é feita uma análise sobre algumas das técnicas mais utilizadas, e é proposto um método caracterizado por duas fases: primeiro detecta os sons impulsivos e, em seguida, a janela encontrada é passada para a parte de identificação. Uma das técnicas mais utilizadas é a aprendizagem supervisionada, onde os modelos são treinados com conjuntos de dados etiquetados (*labeled datasets*).

Com o avançar da tecnologia e a disponibilidade de melhores recursos computacionais, modelos mais complexos, nomeadamente, redes neuronais artificiais, podem ser treinados para se alcançar o estado da arte sob o ponto de vista do desempenho, na classificação de sons impulsivos. Atualmente, existem plataformas que permitem criar, treinar e lançar modelos de *machine learning* e *deep-learning* na nuvem (*cloud-based solutions*), como é o caso da Amazon SageMaker[1], da Microsoft Azure Machine Learning[2], da Google Cloud AI Platform[3], da H2O.ai[4], entre outras. Aquelas plataformas disponibilizam os recursos computacionais necessários para lidar de forma eficiente com conjuntos de dados de grande dimensão. Existem também estruturas/bibliotecas

[1] Amazon SageMaker website, https://aws.amazon.com/pt/sagemaker/

[2] Microsoft Azure Machine Learning website, https://azure.microsoft.com/en-us/products/machine-learning

[3] Google Cloud AI Platform website, https://cloud.google.com/products/ai?hl=pt-br

[4] H2O.ai website, https://h2o.ai/

de *software*, tais como, o TensorFlow[5] (desenvolvido pela Google), o PyTorch[6] (desenvolvido pela Facebook), Apache MXNet[7], e o Scikit-Learn[8], que permitem a criação de algoritmos de *machine learning* e *deep-learning* do zero.

As Redes Neuronais Convolucionais (CNNs) são o tipo de rede neuronal mais adequado atualmente para trabalhar com classificação de imagens. Têm sido também, uma das técnicas mais utilizadas na classificação de sons [8]. A ideia é representar o som através de imagem, e para isso, considera-se a imagem do espectrograma ou MFCCs (Mel Frequency Cepstral Coefficients). Foi demonstrado que este tipo de redes é capaz de obter excelentes resultados na classificação de áudio quando comparado com uma rede neuronal simples (*fully connected*), ou com arquiteturas anteriores de classificação de imagem [14].

Para explorar o espaço de híper-parâmetros, existem atualmente várias técnicas e algoritmos. As abordagens mais conhecidas (e mais fáceis de implementar) são a grid-search e a random-search [2, 11]. Com o aumento do volume de dados e do espaço de híper-parâmetros, a técnica grid-search tem um custo computacional muito elevado [11, 13]. A utilização da técnica random-search é apresentada como alternativa eficiente à grid-search na otimização de híper-parâmetros [7].

Testar uma única configuração de híper-parâmetros em datasets de grandes dimensões pode, nos dias de hoje, facilmente ultrapassar várias horas ou até mesmo vários dias [13]. Existem outras abordagens, mais avançadas, para uma exploração mais eficiente do espaço de híper-parâmetros, como a otimização Bayesiana (Bayesian-optimization) [27] e o HyperBand [17]. Aquelas abordagens aceleram as avaliações das diferentes configurações de híper-parâmetros comparativamente aos métodos exaustivos como o grid-search [2]. O algoritmo Bayesian-optimization é considerado um dos algoritmos que representam o estado da arte na optimização de híper-parâmetros [11, 13].

Nos últimos anos, foram disponibilizadas várias bibliotecas e estruturas que implementam aquelas técnicas. Algumas das bibliotecas mais populares são: Scikit-optimize[9]; Hyperopt[10]; Keras tuner[11]; e Optuna[12]. De uma forma geral, as bibliotecas para otimização de híper-parâmetros suportam-se na definição de uma função objetivo (a minimizar ou maximizar).

[5]TensorFlow website, https://www.tensorflow.org/?hl=pt-br
[6]PyTorch website, https://pytorch.org/
[7]Apache MXNet website, https://mxnet.apache.org
[8]Scikit-Learn website, https://scikit-learn.org/stable/
[9]Scikit-optimize website, https://scikit-optimize.github.io/stable/
[10]Hyperopt website, https://hyperopt.github.io/hyperopt/
[11]Keras Tuner website, https://keras.io/keras_tuner/
[12]Optuna website, https://optuna.org/

A classificação de sons impulsivos é uma área de investigação em pleno desenvolvimento, com vários desafios a serem enfrentados. É necessário lidar com conjuntos de dados desequilibrados, enquanto a quantidade e qualidade dos dados deve permitir manter a robustez do modelo em diferentes condições (e.g., ambiente *indoor*; *outdoor*), e ainda, lidar com o ruído ambiente.

3

Método proposto

3.1 Fundamentos

3.1.1 Eventos acústicos

O conceito de som está relacionado com a propagação de uma onda mecânica acústica, longitudinal, a qual se propaga geralmente de forma esférica. Estas ondas (acústicas) podem ser produzidas por qualquer objeto de produza vibração. Esta representação do som é conhecida como sinal acústico (ou sinal de áudio), para a gama de frequência entre os 20 Hz e os 20 kHz, ou seja, a banda audível.

Um sinal acústico tem vários parâmetros: duração (temporal); comprimento de onda; período; amplitude; e frequência. Todos estes parâmetros devem ser considerados na identificação das características do som.

Um sinal analógico é representado por uma onda contínua que varia ao longo de um período. Existe um número infinito de amostras entre quaisquer duas instâncias de tempo consecutivas. Por outro lado, um sinal digital é uma representação discreta durante um período, e existe um número finito de amostras entre quaisquer duas instâncias de tempo consecutivas.

Para converter um sinal analógico em digital (i.e., um sinal contínuo num sinal discreto), a técnica mais utilizada é a amostragem (*sampling*). Esta técnica consiste em medir a amplitude da onda sonora em intervalos fixos de tempo, e quantificar essas

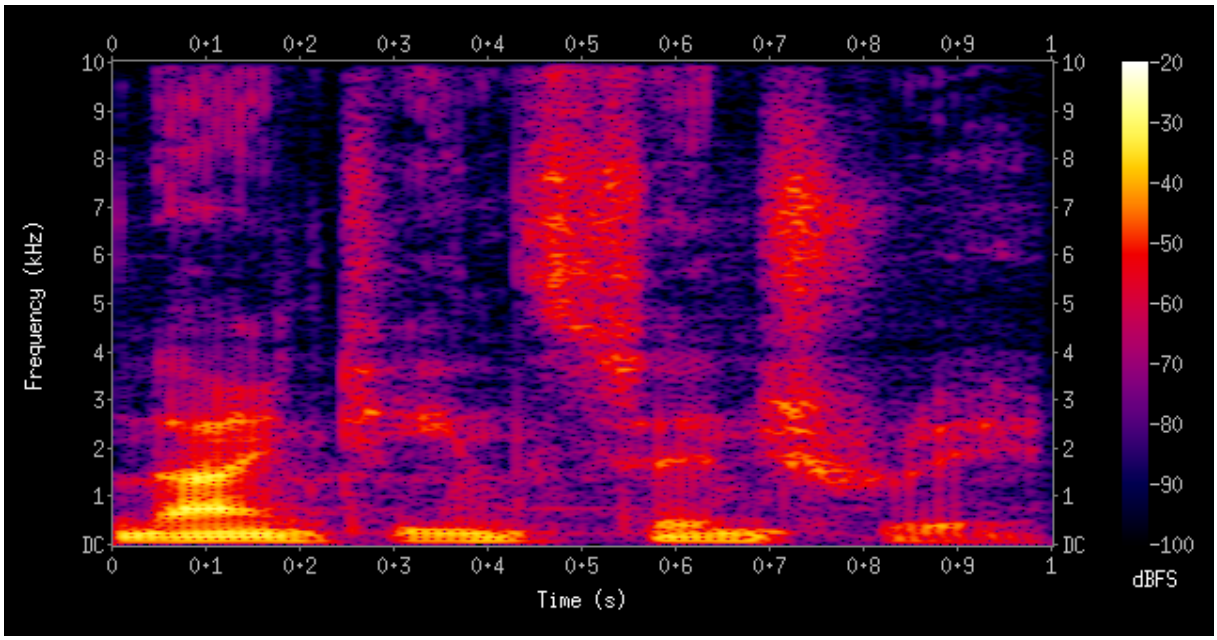

Figura 3.1: Exemplo de espectrograma.

medições como valores digitais. O número de amostras recolhidas por segundo é chamado de "frequência de amostragem" (*sampling frequency*), e é medido em Hertz (Hz). De acordo com o teorema de Nyquist-Shannon, a frequência de amostragem deve ser pelo menos duas vezes maior do que a frequência mais alta presente na onda sonora original, para que a informação seja preservada com fidelidade.

Quando captamos amostras do sinal no domínio do tempo, captamos apenas as amplitudes resultantes. A transformada de Fourier permite decompor um sinal nas suas componentes individuais de frequência e a respetiva amplitude. Ou seja, esta técnica permite converter um sinal no domínio do tempo para o domínio da frequência.

É possível representar o espectro do sinal preservando a informação temporal: resumidamente, a técnica consiste em aplicar o cálculo do método Short-time Fourier Transform, e obtém-se o espectrograma (representação tempo-frequência). Um espectrograma é um gráfico bidimensional entre tempo e frequência, onde as amplitudes são representadas por uma escala de cores. Cada valor do espectrograma representa uma amplitude da frequência num determinado momento. A Figura 3.1[1] mostra um exemplo do espectrograma.

Um som impulsivo, no que diz respeito às suas características acústicas, distingue-se de outros sons (e.g., estacionários; intermitentes; de baixa frequência) pela sua (muito) curta duração, e por um rápido e forte aumento da intensidade do som (*onset* ou *attack*), seguido de um rápido decaimento (*decay* / *release*) que depende do ambiente acústico

[1]Fonte: https://en.wikipedia.org/wiki/Spectrogram

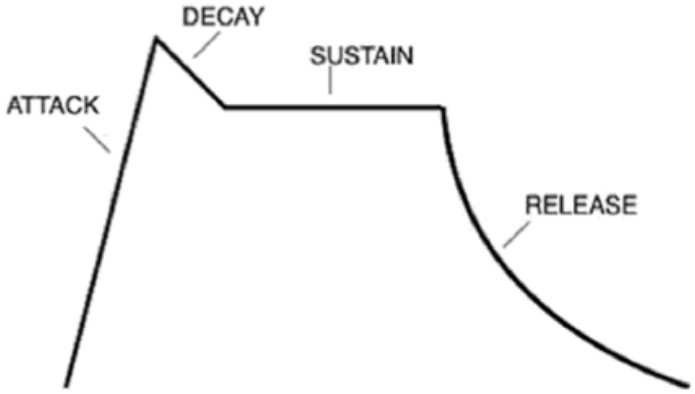

Figura 3.2: Exemplo de representação da envolvente ADSR.

Figura 3.3: Exemplo de forma de onda (direita) e a respetiva envolvente ADSR destacada (esquerda).

onde o som é gerado. Na área da música, uma das formas mais utilizadas para analisar, ou produzir (com sintetizadores), sons de instrumentos musicais, consiste em aplicar uma envolvente ADSR de amplitude ao som [20]. ADSR é o acrónimo de: Attack (ataque), Decay (decaimento), Sustain (sustentação), Release (relaxamento). Estes termos representam as 4 fases que caracterizam o envelope acústico de um som:

- Ataque (*Attack*) – determina a rapidez com que o sinal atinge sua amplitude máxima antes de entrar na fase de decaimento.
- Decaimento (*Decay*) – determina a duração da queda desde o nível de pico até atingir a fase de sustentação.
- Sustentação (*Sustain*) – determina a amplitude do sinal durante todo o tempo entre as fases de decaimento e relaxamento.
- Relaxamento (*Release*) – determina a duração da queda na qual o nível do sinal volta a zero.

A Figura 3.2[2] ilustra um exemplo de representação da envolvente ADSR, e a Figura 3.3[3] ilustra um exemplo de forma de onda e a respetiva envolvente ADSR destacada.

[2]Fonte da imagem: [20].
[3]Fonte: https://pt.wikipedia.org/wiki/ADSR

Num som impulsivo, a fase de sustentação é removida, e as fases de decaimento e relaxamento são unidas. Cada som tem o seu próprio perfil, o qual é influenciado por diversas características. O mesmo som, gerando em ambientes (e espaços) diferentes, também terá um perfil diferente. Além disso, a reverberação do espaço tem uma influência significativa na duração do som impulsivo.

Exemplos de sons impulsivos: disparo de uma arma de fogo; batida de bola numa parede ou numa raquete; batida de martelo; explosão; rebentamento de pneu; entre outros.

3.1.2 Características (*features*) do sinal acústico

Um sinal acústico tem características (*audio features*) específicas que o permitem identificar e descrever. Um som impulsivo pode ser descrito pela sua curta duração, onde a sua energia tem um crescimento rápido inicialmente, seguido de um decaimento que depende do ambiente acústico onde o som é gerado. Cada característica (*audio feature*) representa, essencialmente, uma particularidade específica do sinal acústico, e pode ser medida por técnicas de processamento de áudio, obtendo-se um valor que quantifica aquela característica. Esses valores podem ser utilizados para alimentar o conjunto de dados na criação de modelos de aprendizagem automática, tais como, classificadores de eventos sonoros, entre outros (e.g., processamento da fala, reconhecimento de géneros musicais, etc.). Tipicamente, as características são categorizadas de acordo com vários aspetos [10]:

- Nível de abstração:
  - Alto – identificáveis facilmente pelo ouvido humano;
  - Médio – percetíveis pelo ouvido humano;
  - Baixo – o ouvido humano não as consegue identificar, mas as máquinas conseguem.
- Duração temporal: instantânea; segmentada; global.
- Aspeto musical: propriedades acústicas que incluem o ritmo, timbre, melodia, etc.
- Domínio do sinal: características extraídas do áudio a partir da sua representação no domínio do tempo (*waveform*); no domínio da frequência; e na representação tempo-frequência.

Nos algoritmos de aprendizagem automática tradicionais (e.g., Suport Vector Machines) é feita a extração manual de características (*Hand-picked features / feature engineering*), sendo consideradas as *features* no domínio do tempo (e.g., Root Mean Square, Zero-crossing rate) e também no domínio da frequência (e.g., Band Energy Ratio, Spectral centroid). Nos algoritmos de aprendizagem profunda (e.g., Neural Networks) é feita a extração automática de características, podendo ser consideradas representações não estruturadas do áudio (e.g., mel-spectrogram, MFCC).

Com base na noção geral do que distingue um som impulsivo, são mencionadas em seguida as *audio features* potencialmente relevantes (do conjunto das mais documentadas na literatura e implementadas pelos softwares de processamento de áudio).

Domínio do tempo:

- ***Onset Detect*** – deteção do instante em que se inicia um determinado som. Considera-se uma *feature* potencialmente relevante, pois é uma técnica que pode ser usada para detetar o rápido e forte aumento da intensidade que caracteriza o som impulsivo. A escolha desta *feature* é feita também com base no trabalho relacionado [12].

- ***Root Mean Square (RMS)*** – valor eficaz. É a energia média concentrada num intervalo. O intervalo que seja de um som impulsivo terá um valor de RMS mais alto. A escolha desta *feature* é feita também com base no trabalho relacionado [12].

- ***Amplitude envelope*** – refere-se à variação da amplitude de um som ao longo do tempo [19]. Esta *feature* dá uma ideia aproximada do volume[4]. Pode ser útil no contexto de *onset detection*.

- ***Zero Crossing Rate (ZCR)*** – taxa de cruzamento do valor zero, ou seja, a taxa de mudanças de valor negativo -> zero -> positivo e positivo -> zero -> negativo. Por outras palavras (e considerando a representação da onda sonora no plano cartesiano), é o número de vezes que o sinal cruza o eixo horizontal. Esta característica apresenta bons resultados no reconhecimento da fala, de música, sendo uma característica importante na classificação de sons percussivos [25] (i.e., sons produzidos por instrumentos musicais de percussão, como os pratos e tambores de uma bateria, por exemplo). Alguns sons percussivos podem ser considerados sons impulsivos pois têm um pico de energia muito alto e uma duração muito

[4]Na área do áudio, o conceito de "volume" está relacionado com a perceção subjetiva (i.e., pelo ouvido humano) da amplitude ou intensidade do som, enquanto a amplitude é uma medida objetiva da intensidade física do som (i.e., magnitude de uma onda sonora).

curta. Os instrumentos de percussão são tocados por batidas ou fricção, produzindo sons que fornecem uma sensação de pulsação/impulso à música. No entanto, nem todos os sons percussivos podem ser considerados impulsivos; por exemplo, instrumentos de percussão como maracas, chocalhos, e guizos produzem sons mais suaves e prolongados, não tendo um ataque tão rápido e definido como os instrumentos de percussão de batida. Sons altamente percussivos tendem a ter valores de ZCR mais altos [29].

Domínio da frequência:

- ***Band Energy Ratio (BER)*** – comparação da energia nas bandas de frequência inferior/superior. Os sons impulsivos caracterizam-se por ter um espetro uniforme, mostrando uma ampla gama de frequências. Em comparação com sons contínuos, os sons impulsivos tendem a ter espectros menos concentrados em frequências específicas. A forma exata do espectro depende das características específicas do som impulsivo em questão; uma batida de bola, na modalidade de ténis e padel, tem a energia do espetro mais concentrada nas baixas frequências (som mais abafado).

- ***Spectral Flux*** – verificação das variações na distribuição de energia espectral de um sinal de áudio ao longo do tempo, com o objetivo de encontrar diferenças entre intervalos consecutivos, o que permite detetar o início de um som. Pode também ser definido como uma medida do quão depressa o espetro de um sinal varia [21]. Nos sons impulsivos ocorre uma mudança rápida na distribuição de energia espectral (i.e., variação da energia em diferentes frequências), o que significa que o *Spectral Flux* será alto. A escolha desta *feature* é feita também com base no trabalho relacionado [12].

- ***Spectral Centroid*** – representa o centro de gravidade espectral. Fornece uma ideia de quão "agudo"ou "grave"é um som. Os sons impulsivos tendem a ter componentes de alta frequência significativas.

- ***Spectral Spread*** **(também conhecida por** ***Bandwidth)*** – descreve a dispersão das frequências no espectro de um sinal de áudio. Pode ser usado para diferenciar entre sons ruidosos (alta dispersão espectral) e sons agudos (baixa dispersão espectral) [21].

Representação tempo-frequência:

- ***Spectrogram*** – o espetrograma mostra a variação das frequências do sinal de áudio ao longo do tempo. Podem ser extraídas características tais como, a intensidade de frequência em diferentes intervalos de tempo, e deteção de padrões na imagem do espectrograma.

- ***Mel-spectrogram*** – é um espectrograma onde as frequências são convertidas para a escala "mel". Trata-se de uma escala logarítmica baseada no princípio de que distâncias iguais na escala têm a mesma distância percetiva. Motivação: o ouvido humano tem a perceção do som numa escala logarítmica, ou seja, consegue detetar melhor diferenças em frequências mais baixas do que em frequências mais altas (e.g., deteta facilmente a diferença entre 500 e 1000 Hz, mas dificilmente deteta a diferença entre 10.000 e 10.500 Hz).

  Os ***Mel-Frequency Cepstral Coefficients (MFCCs)*** são calculados[5] a partir do *Mel-spectrogram*, e são essencialmente uma representação compacta das características espectrais do sinal de áudio na escala "mel".

  A principal utilização desta técnica é em reconhecimento de voz [21]. Neste trabalho não é considerada como potencialmente relevante para a deteção de sons impulsivos.

3.1.3 Aprendizagem automática

A aprendizagem automática (*Machine Learning - ML*) é um subconjunto da vasta área da Inteligência Artificial (IA).

O conceito de Inteligência Artificial significa, essencialmente, a capacidade de um computador ou peça de software para reproduzir competências semelhantes às humanas, como é o caso do raciocínio, a aprendizagem, o planeamento e a criatividade. Ou seja, é o desenvolvimento de sistemas computacionais capazes de desempenhar tarefas que normalmente requerem inteligência humana.

O conceito de aprendizagem automática pode ser definido, segundo Arthur Lee Samuel (1901 - 1990), cientista norte-americano, como a capacidade de um sistema computacional "aprender" de forma automática, ou seja, sem ser (explicitamente) programado para isso. Esta capacidade depende de algoritmos que possibilitem o sistema computacional de melhorar automaticamente (os resultados) através da própria experiência [22].

A diferença entre ML e a programação tradicional, é que a aprendizagem automática compensa o desconhecimento do algoritmo com a presença de dados representativos

[5] A informação sobre a forma de cálculo pode ser consultada em [16].

do problema. Em ML, em vez de se escrever a função, é passado um conjunto de dados ao algoritmo, e o próprio algoritmo vai identificar padrões nesses dados, e vai gerar essa função, com a qual se poderá fazer previsões. Esse conjunto de dados é geralmente chamado *dataset*. Resumindo, pode-se dizer que um algoritmo de aprendizagem automática consiste em, a partir de conjuntos de dados (que representam a informação sobre algo), identificar padrões nesses dados de forma a saber dar um resultado sobre novos dados (dados que o algoritmo não conhece porque não foram usados na fase de treino).

A aprendizagem automática está organizada em 4 áreas: aprendizagem supervisionada; aprendizagem não supervisionada; aprendizagem semi-supervisionada; e aprendizagem por reforço.

No problema abordado neste trabalho, pretende-se classificar algo a partir das suas características, portanto, trata-se de um problema de classificação; a deteção e identificação de sons impulsivos é um problema de classificação binária[6], em que o sistema deve fazer a distinção entre batidas de bola e ruído. Como tal, vai ser considerada a aprendizagem supervisionada.

Na aprendizagem supervisionada, o conceito de supervisão representa essencialmente a atribuição de uma "etiqueta" (classe) a cada instância (ou "exemplo") do conjunto de dados. Esses dados são representados em forma de matriz – matriz de dados, X – em que, cada linha da matriz representa um registo, caracterizado por *features* (características) indicadas em cada coluna da matriz. Existe também um vector de etiquetas, representado por Y, que indica a classe a que pertence cada registo (linha) da matriz de dados. Nas tarefas de classificação são usadas etiquetas discretas.

3.1.4 Algoritmos de classificação

Na área da aprendizagem supervisionada existe um conjunto vasto de algoritmos para classificação e também para regressão[7]. Embora o foco deste capítulo seja o conjunto de algoritmos utilizados em problemas de classificação, alguns deles podem ser também utilizados em problemas de regressão.

De seguida são apresentados alguns algoritmos, entre os mais utilizados (em trabalhos relacionados) e documentados na literatura, desde os mais básicos aos mais complexos (sob o ponto de vista do espaço de híper-parâmetros).

[6]Classificação binária – quando existem apenas duas classes nos dados. Um exemplo não pode pertencer, simultanemente, a mais do que uma classe.

[7]Nos algoritmos para resolver problemas de regressão, o objetivo não é atribuir uma classe aos dados, mas sim, prever/estimar valores numéricos.

- ***K-Nearest Neighbor* (K-NN)** – Pode ser usado tanto como algoritmo de aprendizagem supervisionada como não-supervisionada. Na aprendizagem supervisionada, pode ser utilizado em problemas de classificação e também de regressão. O termo "neighbor"(vizinho) refere-se ao(s) exemplo(s) do *dataset*. O algoritmo utiliza a informação dos K vizinhos mais próximos no espaço de características (*features*). Cada eixo/dimensão do espaço representa uma *feature*. Para cada exemplo (do conjunto de teste), o algoritmo determina a sua posição correspondente no espaço de características, e depois identifica os K vizinhos (do conjunto de treino) mais próximos nesse espaço, através da medição da distância (e.g., distância Euclidiana). Para encontrar esses vizinhos mais próximos, são usados algoritmos de indexação. Por fim, o algoritmo identifica a classe mais representada por esses K vizinhos, e elege essa classe para classificar o exemplo.

  Híper-parâmetros:

  - K: número de vizinhos a considerar. Valor inteiro positivo.
  - Pesos (*weights*): pode-se dar aos K vizinhos mais próximos igual prioridade, ou decidir os seus pesos com base na distância ao ponto; quanto maior for a distância, menor será a ponderação.
  - Algoritmo de indexação: é usado para mapear os vizinhos mais próximos. Para evitar a necessidade de se calcular a distância entre todos os exemplos do *dataset*, são usados algoritmos de indexação baseados em árvore (*tree-based indexing algorithms*) tais como *kd-tree* e *ball tree*. O algoritmo *kd-tree* particiona os dados em eixos cartesianos, e o algoritmo *ball tree* particiona-os em híper-esfera aninhada (*nested hyper-sphere*). Quando se tem um grande número de dimensões, o algoritmo *ball tree* é mais eficiente que o *kd-tree* [2].
  - Métrica do cálculo da distância: usada para calcular a distância entre exemplos. A métrica pode ser, por exemplo, a distância Euclidiana[8] ou a distância de Manhattan[9].

- **Árvore de Decisão *(Decision Tree)*** – Como o próprio nome sugere, é uma "árvore", portanto, o processo de previsão/classificação de um exemplo começa com o nó raiz, que representa uma das *features*. Com base no valor dessa *feature*, é selecionado o próximo nó, e assim por diante (analogamente a um encadeamento de declarações *if-else*), até se atingir um nó folha, o qual representará a previsão

[8] A definição formal da distância Euclidiana pode ser consultada em: https://xlinux.nist.gov/dads/HTML/euclidndstnc.html

[9] A definição formal da distância de Manhattan pode ser consultada em: https://xlinux.nist.gov/dads/HTML/manhattanDistance.html

para aquele exemplo. Existem vários algoritmos para selecionar as *features* a representar nos vários nós, calculando a importância/prioridade de cada uma (das *features*). Um dos problemas das Árvores de Decisão é que quando aumenta a complexidade da árvore, há uma grande tendência para o modelo se ajustar demasiado ao conjunto de treino (*overfitting*). Alguns dos híper-parâmetros podem ajudar a reduzir essa complexidade; o objetivo será "podar a árvore" (reduzir a profundidade), utilizando os híper-parâmetros *Depth of Tree*, e *Minimum Sample Leaf*.

Híper-parâmetros:

- Algoritmo: decide a importância/prioridade das *features* e, portanto, a sua ordem na estrutura da árvore.
- Profundidade da árvore (*Depth of Tree*): define a profundidade da árvore.
- Divisão mínima de amostra (*Minimum Sample Split*): define o número mínimo de amostras necessárias para dividir um nó interno.
- Número mínimo de amostras no nó folha (*Minimum Sample Leaf*): define o número mínimo de amostras no nó folha.

- **Máquina de suporte vetorial *(Support Vector Machine – SVM)*** – Pode ser utilizado em problemas de classificação e também em problemas de regressão. A ideia por trás do algoritmo consiste em (na fase de treino) determinar um plano hiperdimensional (híper-plano) que otimize/maximize a separação entre classes distintas. Essa separação é chamada de "margem". Cada exemplo que se pretende classificar, é representado por um ponto no híper-plano. As coordenadas desse ponto são as *features*. Os vetores de suporte (*support vectors*) são pontos que estão mais próximos do híper-plano e influenciam a sua posição e a orientação. Usando esses vetores de suporte, maximiza-se a margem do classificador. Para determinar o híper-plano, é usada a técnica dos *multiplicadores de Lagrange*[10].

  Na tarefa de classificação binária, o algoritmo desenha um híper-plano que separa os pontos em 2 lados (i.e., ficam de um lado do híper-plano se pertencerem a uma classe, e do outro lado se pertencerem à outra classe).

  No caso mais simples, para classes linearmente separáveis onde conseguimos visualizar em duas dimensões, o algoritmo tenta separá-las por uma linha reta. Portanto, o algoritmo define a reta, calcula a distância da reta aos pontos (de cada

[10] A técnica dos *multiplicadores de Lagrange* é uma forma de resolver problemas de otimização com restrições. Permite encontrar o máximo ou o mínimo de uma função multivariável f(x, y, ...) quando há alguma restrição sobre os valores de entrada que podem ser usados.

classe) mais próximos dela, e tenta maximizar essa margem, de forma que a reta fique o mais distante possível das classes. A posição da reta tem influência na capacidade de generalização do modelo.

Para os casos em que o conjunto de dados não é linearmente separável, existem variações do SVM, mais concretamente, o conceito de *Kernel*, que permite aplicar o SVM a conjuntos de dados não linearmente separáveis, variando uma função *kernel*. É importante conhecer-se as características dos dados do *dataset* (por exemplo, se são linearmente separáveis ou não), para se poder escolher a função *kernel* mais adequada.

Híper-parâmetros:

- *kernel*: permite encontrar a função de mapeamento (do *dataset*) mais adequada ao problema. Os *kernels* mais usados e implementados nas bibliotecas (e.g., Scikit-Learn) são: *linear kernel, RBF (Radial Basis Function) kernel, poly (polynomial kernel), Gaussian kernel*, e *sigmoid kernel*. Os não lineares têm um custo computacional mais elevado. O utilizador pode também definir o seu próprio *kernel*.
- C: é o inverso do peso dado ao termo de regularização na função de custo. Serve para indicar se a "margem" deve ser mais rígida (C maior) ou mais suave (C menor), equilibrando a precisão do treino de acordo. Um aumento de C resulta em margens menores e numa precisão maior do treino, e vice-versa.
- Gamma: este híper-parâmetro (γ) pode ser visto como sendo o coeficiente do *kernel*, e permite definir a influência dos pontos na fase de treino.
- Grau (*Degree*): usado apenas no *kernel polynomial*. Um valor maior significa uma fronteira de decisão mais flexível. O valor 1 resulta num *kernel* linear.

- **Rede neuronal perceptrão multi-camada *(Multi Layer Perceptron - MLP)*** – Uma rede neuronal está organizada em camadas de neurónios[11] interligados, onde cada neurónio processa informações[12] e passa os resultados para a próxima camada. Os dados de entrada, ou seja, as características (*features*), são inseridos na rede pela camada de entrada (*input layer*); passam pelas camadas intermédias/escondidas (*hidden layers*) onde são ponderados e transformados; o resultado é apresentado na última camada (*output layer*). Os parâmetros (peso (*weight*) e o

[11] No contexto de uma rede neural, um neurónio, também chamado de nó ou perceptrão, é a unidade de processamento mais fundamental. Possui uma ou mais ligações de entrada ponderadas, uma função de transferência que combina as entradas de alguma forma, e uma ligação de saída.

[12] Resultado da camada anterior multiplicado por pesos.

bias) são ajustados para otimizar a qualidade do resultado. O processo de aprendizagem ocorre através de vários ciclos de passagem, pela rede neuronal, de todos os exemplos do *dataset*. Cada ciclo de passagem corresponde a uma época (*epoch*), e cada época é composta por várias iterações. Uma iteração corresponde à passagem pela rede de um número pré-definido (*batch size*) de exemplos, resultando na atualização dos pesos da rede.

Durante a fase de treino ("aprendizagem") do algoritmo, a atualização dos pesos da rede é feita através do processo de retropropagação (*backpropagation*): após a passagem dos dados pela rede (para gerar uma previsão), o resultado da previsão é apresentado na última camada (*output layer*), e é calculada a diferença entre a previsão obtida e o valor real (o erro); essa diferença é propagada de volta pela rede, calculando-se o gradiente do erro em relação aos pesos; o algoritmo de otimização ajusta os pesos com base nesse gradiente, com o objetivo de minimizar o erro.

Existe um vasto número de híper-parâmetros. São apresentados, de seguida, alguns dos mais importantes:

- Número de camadas (*Number of Layers*): cada camada escondida é responsável por aprender características específicas dos dados. Adicionar camadas aumenta a profundidade da rede neuronal e também a capacidade de aprender *features* mais complexas. As camadas (escondidas) podem ajudar a discriminar melhor os dados, o que pode levar a uma melhor separação entre classes. Por outro lado, ao adicionar camadas, há um risco maior de *overfitting*, especialmente se a quantidade de dados para treino é limitada. Camadas adicionais podem levar a que o modelo se ajuste demais aos dados de treino, e não generalize bem para novos dados (dados de teste).
- Número de neurónios por camada (*Number of Nodes*): a escolha do número de nós em cada camada depende do tipo de problema e das características do *dataset*; cada camada escondida pode ter um número diferente de nós; a camada de entrada tem um número de nós igual ao número de *featutes* do *dataset*; a última camada possui os nós que são usados para representar os valores da classe.
- Tamanho do *batch* (*Batch Size*): é o número de exemplos do conjunto de treino usado em cada iteração.
- Função de activação (*Activation Function*): é usada para introduzir não-linearidade em cada nó. A função de ativação em cada camada da rede define como é que os neurónios são ativados. Algumas das funções mais utilizadas são: a

ReLU (Rectified Linear Unit), a sigmoide, e a tangente hiperbólica. Na camada de saída, para problemas de classificação binária, normalmente é usada a função sigmoide.

- Função de perda (*Loss Function*): é escolhida com base na saída, independentemente do tipo de problema (classificação binária, classificação multiclasse, regressão). Existem também outros fatores a considerar; por exemplo, usar a ativação sigmoide na última camada e a função de perda quadrática pode resultar em lentidão na aprendizagem da rede. A função de perda tem também híper-parâmetros internos que podem ser ajustados.
- Algoritmo de otimização (*Optimizer*): é uma função ou algoritmo que modifica os atributos da rede neuronal (tais como, os pesos e a taxa de aprendizagem), para ajudar a melhorar o desempenho do modelo. O optimizador *Adam* é um dos mais utilizados, mas existem vários, tais como, a descida de gradiente com momento, ou o *RMSprop*.
- Taxa de aprendizagem (*Learning rate*): determina o tamanho dos passos que o algoritmo de otimização dá durante a fase de treino. Uma taxa de aprendizagem alta pode acelerar o processo, mas também pode levar a maiores oscilações e dificuldade em atingir a convergência. Por outro lado, uma taxa de aprendizagem baixa pode levar a um processo de treino mais lento, mas mais estável.
- Número de épocas (*Number of Epochs*): aumentar o número de épocas permite que o modelo continue a ajustar os pesos dos nós ao longo do tempo, o que pode levar a um melhor ajuste dos parâmetros. Mas se o número de épocas for muito alto, há o risco de o modelo se ajustar demasiado ao conjunto de treno e não generalizar bem com novos dados (*overfitting*). Na maioria dos casos, o número de épocas não precisa de ser otimizado, bastando simplesmente usar a estratégia de *early stopping* para parar quando não houver nenhuma melhoria no desempenho do modelo.
- Regularização: é uma técnica que permite eliminar a informação que reduz o desempenho do modelo. O objetivo é evitar o *overfitting*. Existem vários tipos de regularização, entre eles: *Dropout*, *Lasso* e *Ridge*, e *early stopping*. A técnica de *Dropout* consiste em desligar aleatoriamente alguns neurónios em cada iteração durante a fase de treino; essa "remoção" aleatória de neurónios obriga, de certa forma, a uma regularização estocástica, evitando que os neurónios dependam excessivamente uns dos outros e, assim, reduzindo o *overfitting*. As regularizações de *Lasso* (ou norma L1) e *Ridge* (ou norma

L2) penalizam os pesos associados a certas características, para reduzir o seu valor. O *early stopping* consiste em interromper o processo de treino da rede quando o erro (*loss function*) da validação atingir o mínimo.

3.1.5 Exploração do espaço de híper-parâmetros

Nos algoritmos de aprendizagem automática, as variáveis podem ser categorizadas em dois tipos:

- **Parâmetros**: são aqueles que o algoritmo ajusta de acordo com o conjunto de dados (*dataset*) fornecido. Considerando, por exemplo, uma Rede Neuronal Artificial, são considerados parâmetros os pesos associados a cada neurónio, entre outros.
- **Híper-parâmetros**: são aqueles ajustados diretamente pelo utilizador, antes de iniciar o processo de treino do algoritmo. Considerando, por exemplo, uma rede neuronal, são considerados híper-parâmetros: a taxa de aprendizagem, o número de camadas da rede, o número de neurónios por camada, entre outros. O ajuste destes híper-parâmetros pode ser baseado nas características do *dataset*, e também na capacidade do algoritmo para aprender [2].

Independentemente do (bom) conhecimento que se possa ter dos métodos de aprendizagem automática para classificação, é praticamente impossível prever, à partida, a configuração dos híper-parâmetros (desses métodos) que maximize o desempenho do modelo. A estratégia de pesquisar manualmente e iterativamente todas as configurações possíveis de valores, representa um custo demasiado elevado, sabendo que existem algoritmos com bons resultados (e.g., redes neuronais), porém com espaços de pesquisa de grande dimensão.

Para ajustar os híper-parâmetros de um método de aprendizagem automática, é importante conhecer bem esse algoritmo e a forma como os híper-parâmetros afetam o seu desempenho. Mesmo quando são usados métodos específicos para otimização de híper-parâmetros, é importante definir um intervalo de pesquisa adequado [2].

Relativamente aos valores possíveis que um híper-parâmetro pode assumir, eles podem ser definidos por uma distribuição discreta[13] ou uma distribuição contínua[14]. Tomando como exemplo o algoritmo *Support Vector Machine* (SVM), o híper-parâmetro

[13]Uma distribuição discreta é um conjunto de valores onde cada valor tem uma distância finita positiva para o próximo valor.

[14]Uma distribuição contínua é um conjunto de infinitos valores possíveis situados entre dois números reais.

kernel (que pode ter o valor *rbf, sigmoid, linear*, etc.) tem uma distribuição discreta, e o híper-parâmetro *C* tem uma distribuição contínua.

De seguida são abordadas algumas das técnicas mais conhecidas na exploração do espaço de híper-parâmetros dos algoritmos de aprendizagem automática.

- **Grid Search** (Pesquisa em Grelha): é especificado (previamente) o conjunto de valores a testar para cada híper-parâmetro, e o desempenho do modelo é avaliado para cada combinação possível de valores, sendo que cada combinação representa uma configuração. O método consiste, portanto, em iterar sobre todas as configurações. Por ser um método de pesquisa exaustiva (abordagem "força-bruta"), é computacionalmente pesado, e demorado, mas é útil para espaços de pesquisa pequenos e com uma distribuição discreta.

- **Random Search** (Pesquisa Aleatória): esta técnica envolve a seleção aleatória de configurações de híper-parâmetros dentro do espaço de pesquisa previamente definido. O número de configurações a serem testadas é definido pelo utilizador, e esta é, teoricamente, uma das vantagens desta técnica comparativamente à *grid search*: o número de configurações testadas pode ser muito menor (e independente do tamanho do espaço de pesquisa), e ainda assim, existem boas possibilidades de ser encontrada uma configuração com bons resultados. A seleção de configurações para teste é feita aleatoriamente, e não são considerados os resultados dos testes anteriores.

  Neste trabalho, não é utilizada esta técnica.

- **Bayesian-optimization** (Otimização Bayesiana): ao contrário das técnicas *grid search* e *random search*, o algoritmo de otimização Bayesiana considera os resultados dos testes anteriores, adaptando-se (com base nesses testes) para explorar de forma eficiente o espaço de híper-parâmetros. Para tal, baseia-se em modelos probabilísticos, tais como, o processo Gaussiano.

  Na otimização Bayesiana, o processo Gaussiano é utilizado para modelar a função objetivo[15] (*objective function*), onde essa função é assumida como uma concretização da distribuição Gaussiana (e onde as previsões seguem uma distribuição normal) [2], e encontrar os limites de confiança superior (*upper confidence bound*) e inferior (*lower confidence bound*) da função.

- **HyperBand**: trata-se de uma técnica que combina busca aleatória e eliminação sucessiva (*successive halving*) das configurações com desempenho insatisfatório. Neste trabalho, não é utilizada esta técnica.

[15] A função objetivo representa o algoritmo de classificação.

3.1.6 Validação do modelo

O modelo deve ter a capacidade de generalizar, ou seja, classificar corretamente dados não vistos anteriormente (i.e., dados que não foram utilizados para treinar o modelo). Para tal, não devem ser utilizados todos os dados disponíveis na fase de treino. Deve ser utilizada (apenas) uma porção dos dados como conjunto de treino (sendo esse o conjunto utilizado na fase de aprendizagem/treino do modelo), e outra porção como conjunto de teste, utilizado da fase de avaliação do modelo aprendido. Por vezes, quando os dados o permitem e os modelos assim o exigem, o conjunto de treino é ainda dividido num subconjunto de validação, o qual serve essencialmente para "afinar" parâmetros do modelo, durante a fase de treino.

Por fim, o conjunto de dados de teste é utilizado no processo de classificação, para avaliar a qualidade/desempenho do modelo. Para que essa avaliação seja confiável, todos os conjuntos de dados (de treino, de validação, e de teste) têm de ser disjuntos. É necessário ter também em consideração a possibilidade de o conjunto de teste ter exemplos mais fáceis de classificar comparativamente ao conjunto de treino, o que resultará numa avaliação enganadora (potencialmente otimista) do desempenho do modelo. A estratégia de validação cruzada (*cross-validation*) permite evitar este problema.

Quando o modelo se ajusta demasiado ao conjunto de treino, ocorre aquilo a que se chama, sobre-aprendizagem (*overfitting*); isso leva a que o modelo produza bons resultados no processo de aprendizagem, e mau desempenho na capacidade de generalização (com o conjunto de teste). Portanto, para não comprometer o desempenho do modelo, deve ser evitado o *overfitting*. Para tal, existem várias estratégias, entre as quais a regularização.

A estratégia de validação cruzada com K partições (*K-fold cross-validation*) permite avaliar a capacidade de generalização de um classificador. A ideia consiste em dividir o *dataset* numa parte para treino e outra parte para teste, e depois, fazer várias amostragens dos dados em K partições. Desta forma, ter-se-á conjuntos de treino e de teste diferentes em cada uma das K validações. No final, o desempenho do modelo é aferido através da média daquelas K validações.

3.1.7 Avaliação do modelo

No contexto de classificação binária, existem várias métricas para avaliar o desempenho do modelo. A mais intuitiva e, bastante utilizada, é a taxa de sucesso (*accuracy*), e é calculada da seguinte forma:

- **Taxa de sucesso (*Accuracy*)** = é o quociente da divisão do número de previsões corretas pelo número total de previsões. Esta métrica representa a probabilidade de acertos do modelo (percentagem de exemplos bem classificados) independentemente da classe. Apesar de ser uma métrica muito utilizada, ela pode não ser suficiente, e em alguns casos o seu resultado pode até induzir em erro, sobretudo quando o *dataset* é desequilibrado[16]. Neste trabalho, pela própria natureza dos dados, é esperado que possa existir uma quantidade maior de exemplos da classe que representa o ruído, comparativamente à classe que representa a batida de bola. Quando se tem um *dataset* desequilibrado, a utilização da taxa de sucesso (*accuracy*) como medida de desempenho não é a melhor opção, porque se o modelo for mau e classificar erradamente todos os exemplos do "dataset" como sendo da classe maioritária, ele terá ainda assim uma alta taxa de sucesso.

Além da taxa de sucesso, existem outras métricas, que permitem verificar a distribuição de acertos em cada classe. Com a matriz de confusão consegue-se extrair 4 conceitos: verdadeiros positivos (VP); falsos positivos (FP); verdadeiros negativos (VN); e falsos negativos (FN). Esses conceitos permitem obter várias métricas (as quais se podem adequar a determinado tipo de problemas), entre as quais:

- **Precisão (*Precision*)** = VP / (VP + FP). Esta métrica diz-nos a percentagem de exemplos que o modelo classificou como sendo de uma determinada classe e que pertencem de facto a essa classe. É especialmente útil quando se pretende reduzir ao máximo os falsos positivos. Por exemplo: num algoritmo que classifica se determinados vídeos são ou não adequados para crianças, querer-se-á que esse algoritmo tenha uma alta Precisão.

- **Cobertura (*Recall*)** = VP / (VP + FN). Esta métrica diz-nos, de todos os verdadeiros positivos, a percentagem de exemplos que foram corretamente classificados. É especialmente útil quando se pretende reduzir ao máximo os falsos negativos. Por exemplo: em algoritmos para identificar doenças.

Em alguns contextos será mais relevante considerar a Precisão, noutros a Cobertura, e noutros, ambas.

- ***F1 Score*** = 2* [(Precisão * Cobertura) / (Precisão + Cobertura)]. Esta métrica corresponde à média harmónica entre a Precisão e a Cobertura. É útil em problemas

[16]O *dataset* diz-se desequilibrado quando a distribuição de exemplos pelas classes é desigual [5]. Esse desequilíbrio pode resultar na construção de um modelo enviesado que reconhece melhor as classes em maioria.

onde o *dataset* é desequilibrado, ou noutros casos em que métricas mais simples possam dar resultados errados [18].

- **Curvas ROC (*Receiver Operating Characteristics*)**. Esta métrica permite ver quão bem o modelo consegue distinguir entre duas classes. Uma curva ROC representa graficamente a relação de dois parâmetros: a taxa de verdadeiros positivos (VP / (VP + FN)); e a taxa de falsos positivos (FP / (FP + VN)). A **AUC (*Area Under the ROC Curve*)** é uma maneira de simplificar a análise da curva ROC, resumindo a curva num único valor [31].

A taxa de sucesso (*accuracy*) pode também ser calculada a partir da matriz de confusão, da seguinte forma: (VP + VN) / (VP + FP + VN + FN).

3.2 Metodologia

O sistema para identificação automática de sons impulsivos é implementado com foco na exploração do espaço de híper-parâmetros, na procura dos valores que otimizem o desempenho de todo o processo de classificação automática de sons impulsivos.

O conceito de híper-parâmetro (HyP) refere-se, em geral, aos valores escolhidos para configuração de determinado algoritmo a executar num processo de aprendizagem automática [2]. Esse algoritmo tem como *input* um *dataset* (os dados) que o algoritmo processa visando aprender (descobrir) padrões contidos nesses dados. O *dataset* é, portanto, a base (a "matéria-prima") de todo o processo.

Neste trabalho propõe-se estender o conceito de HyP de modo a englobar também a construção do próprio *dataset*. Tem-se então, 2 conjuntos de HyP: a) os usados para geração do *dataset* (HyP_data), e b) os usados pelo algoritmo que processa o *dataset* visando aprender a partir desses dados (HyP_learn); de modo mais formal, tem-se:

$$\text{HyP} = \{\text{HyP\_data}, \text{HyP\_learn}\} \tag{3.1}$$

Neste trabalho, os dados representam excertos de áudio, pelo que HyP_data separa-se ainda em dois sub-conjuntos: a) os que estão ligados à amostragem do sinal de áudio, e como tal, são "globais"a todo o restante processamento (HyP_data_global), e b) os que são usados na extração de características (*features*) desse sinal de áudio (HyP_data_feature); tem-se então:

$$\text{HyP\_data} = \{\text{HyP\_data\_global}, \text{HyP\_data\_feature}\} \tag{3.2}$$

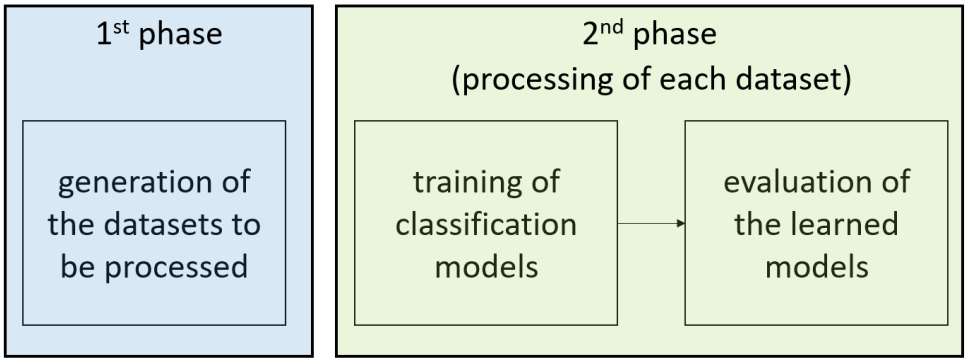

Figura 3.4: Esquema da implementação em duas fases distintas

Os esquemas de HyP (HyP_data_scheme e HyP_learn_scheme) a considerar, são definidos e representados formalmente numa estrutura de dados persistente e separada do código do sistema, contendo o intervalo de valores dos HyP. Assim, o utilizador pode alterar aquele intervalo de valores, e reiniciar o sistema, sem a necessidade de alteração do código. A estrutura formal dos esquemas deve ser sempre mantida, para que a informação (neles contida) possa ser processada de forma genérica pela aplicação.

A implementação contempla duas fases distintas, conforme mostrado na Figura 3.4. Na primeira fase são gerados os (diferentes) datasets considerando os HyP_data e guardando-se os parâmetros que deram origem a cada *dataset*. Na segunda fase é feito o processamento de cada dataset (gerado anteriormente) usando os HyP_learn. Com esta metodologia pretende-se estudar o impacte de diferentes características do *dataset* no desempenho do modelo. Como tal, é essencial que os *datasets* sejam gerados automaticamente e com código genérico.

O processo de geração dos *datasets* engloba um conjunto de etapas, conforme pode ser descrito pela Figura 3.5; nos subcapítulos seguintes são explicadas em detalhe cada uma das etapas.

3.2.1 Recolha de dados (*video repository*)

Os dados utilizados na construção dos *datasets* provêm de vídeos de competições (profissionais ou amadoras) de eventos desportivos, nomeadamente, ténis e padel. No momento da realização deste trabalho, estão a ser utilizados vídeos de jogos de padel apenas.

Geralmente, o áudio é extraído dos vídeos com formatos de áudio e vídeo compactados. Sempre que for possível o áudio ser captado separadamente, então ele deverá ser gravado no formato *raw* ou *wav*, pois são formatos de arquivo sem perdas e, como tal,

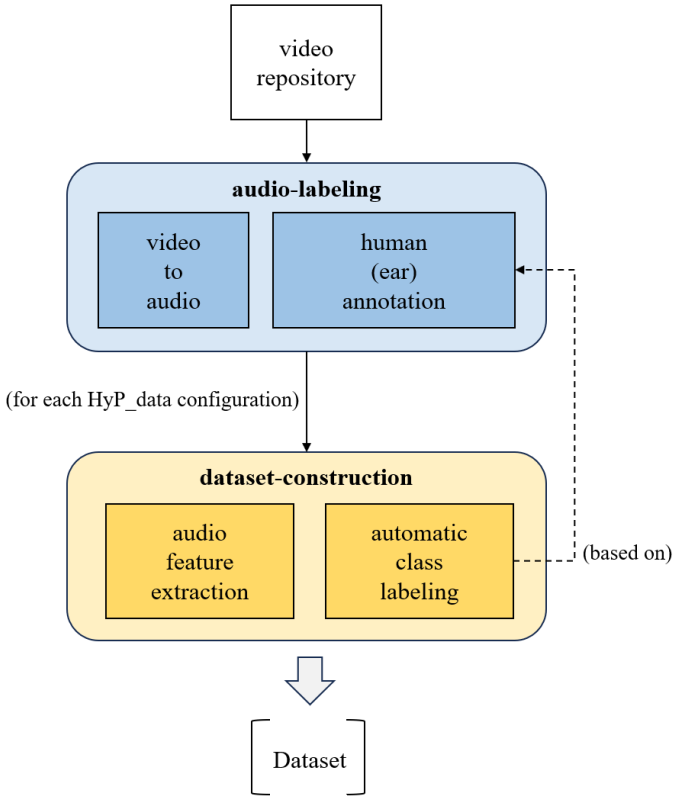

Figura 3.5: Processo de construção do *dataset*.

a representação mais próxima do áudio original [6], preservando transientes e largura de banda.

Existem diversos conjuntos de dados (*datasets*) de áudio disponíveis na internet para fins de pesquisa e desenvolvimento na área da aprendizagem automática e processamento de sinais, em aplicações como, processamento de fala, reconhecimento de voz, classificação de sons ambientais, entre outros. Alguns exemplos: URBAN-SED[17]; ESC-50[18]; DCASE 2016[19]; Freesound[20]; e MIVIA Audio Events Dataset[21]. Considerando os sons impulsivos em particular, os dados mais comuns disponíveis são de sons de disparos de arma (*gun shots*). Assim, não sendo fácil encontrar conjuntos de dados específicos de áudio de jogos de padel disponíveis publicamente, é necessário criar manualmente o próprio conjunto de dados, através da captação directa de vídeos, ou a recolha de vídeos já gravados e disponíveis publicamente – no YouTube[22] por exemplo.

[17]URBAN-SED website, http://urbansed.weebly.com/
[18]ESC-50 website, https://github.com/karolpiczak/ESC-50
[19]DCASE 2016 website, https://dcase.community/challenge2016/index
[20]Freesound website, https://freesound.org/
[21]MIVIA Audio Events Dataset website, https://mivia.unisa.it/datasets/audio-analysis/mivia-audio-events/
[22]YouTube website, www.youtube.com

No caso de ser feita a captação manual de vídeos, é utilizada câmara com microfone externo, e os sinais de áudio são capturados a uma alta frequência de amostragem (*sampling frequency*) de forma a preservar-se a fidelidade da natureza impulsiva do som. No estudo [30] por exemplo, o áudio foi capturado a uma frequência de amostragem de 204,8 kHz.

O conjunto total dos vídeos deve representar variabilidade das condições com efeito nas características acústicas. Ou seja, ambientes interiores e exteriores (*indoor* e *outdoor*), e diferentes condições, tais como, humidade no ar, nível de ruído ambiental, e diferentes distâncias do microfone que correspondem a diferentes níveis da relação sinal-ruído.

3.2.2 Extração de características do áudio (*audio feature extraction*)

As características são extraídas do áudio a partir da sua representação em vários domínios: no domínio do tempo, domínio da frequência, e na representação tempo-frequência.

É feito um varrimento sobre o áudio com uma janela temporal de dimensão fixa – 'event length' – de acordo com a duração do evento sonoro que se pretenda considerar[23]. Esse varrimento é feito em várias iterações, onde em cada iteração a janela desliza um número específico de amostras – 'number of shifted samples'. A janela é dividida em segmentos – 'number of segments' – e a extração das características é feita sob as amostras abrangidas por cada segmento (e não sob o total de amostras da janela), de forma a preservar o contexto temporal [12] e as fases que caracterizam a envolvente ADSR. O número de amostras abrangidas por cada segmento depende também da frequência de amostragem do áudio – 'sampling frequency'. Cada janela de evento corresponde a um exemplo do *dataset*. A Figura 3.6 ilustra o processo de varrimento sobre o áudio para extração de características, com um exemplo de 'number of segments' = 7.

Os conceitos referidos acima ('event length'; 'number of shifted samples'; 'number of segments'; 'sampling frequency') estão ligados à amostragem do sinal de áudio e são "globais"a todo o restante processamento. Como tal, compõem o conjunto de HyP_data_global.

O conjunto de HyP_data_feature é composto pelas *audio features* (Onset Detect; Root Mean Square; Band Energy Ratio; Spectral flux; etc.) e seus (eventuais) parâmetros

[23] A reverberação do espaço tem uma grande influência na duração do som impulsivo. Dependendo do ambiente (e.g., *indoor* ou *outdoor*), os sons propagam-se de forma diferente. Nesse sentido, a duração do evento ('event length') deve ser ajustada de acordo.

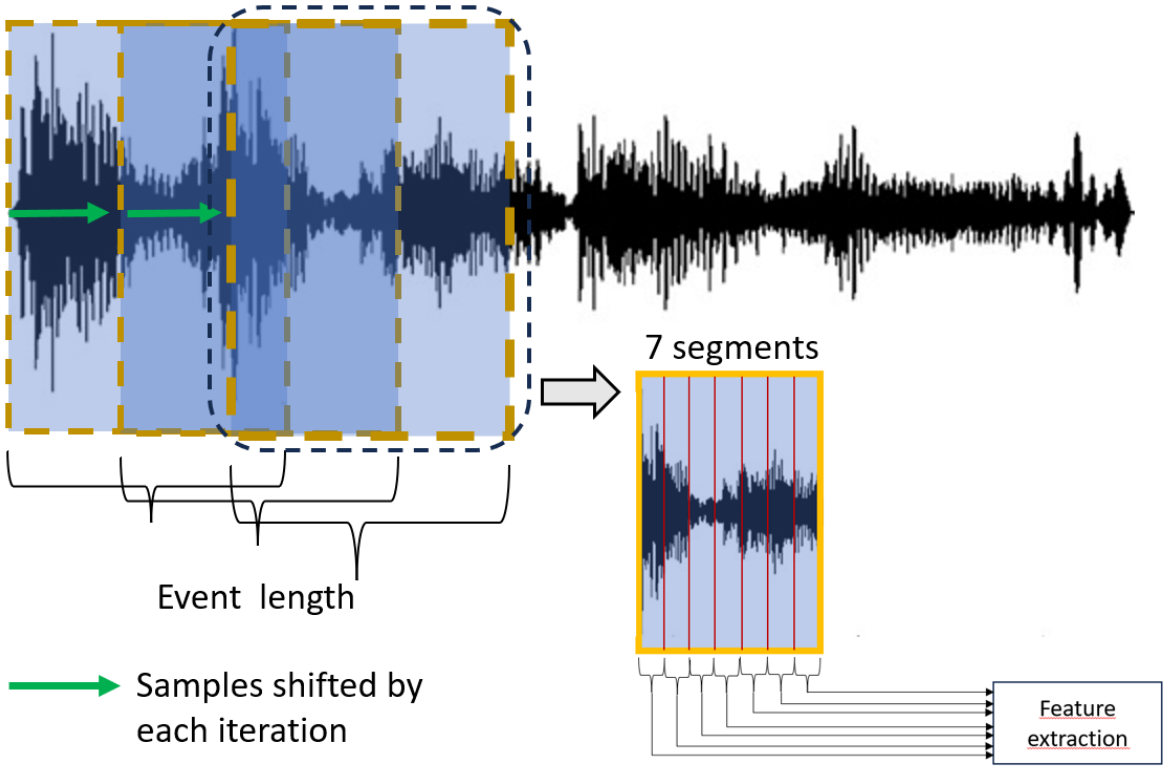

Figura 3.6: Varrimento sobre o áudio para extração de características.

próprios.

3.2.3 Anotação de eventos sonoros (*audio-labeling*)

O processo de anotação de eventos sonoros inclui a extração de áudio (a partir do vídeo) e a anotação manual de eventos sonoros, relacionados com os sons impulsivos. Portanto, cada anotação representa a percepção humana, baseada na audição, de um som impulsivo específico. Dessa forma, o processo de aprendizagem automática visa construir modelos de classificação que alcancem um detalhe de percepção semelhante ao do ouvido humano.

A anotação (manual) baseada no ouvido humano (*human ear annotation*) é auxiliada por software de edição de áudio, como por exemplo o Adobe Audition[24] ou o Audacity[25]. Cada anotação consiste em anotar, em ficheiro, o intervalo de tempo (primeira e última amostra) em que o som é percebido pelo ouvido humano. Esta anotação manual é usada para gerar as etiquetas (vetor de classes) do *dataset* (*automatic class labeling*) que

[24] Adobe Audition website, https://www.adobe.com/pt/products/audition.html, acedido em Fevereiro de 2024.
[25] Audacity website, https://www.audacityteam.org/, acedido em Fevereiro de 2024.

serve de *input* para o processo de aprendizagem automática.

Portanto, tem-se aprendizagem supervisionada guiada pela relação entre o ouvido humano e a construção do *dataset*.

3.2.4 Etiquetagem automática dos dados (*automatic class labeling*)

A etiquetagem automática dos dados é realizada com base na anotação de eventos sonoros (*audio-labeling*). Para cada exemplo do conjunto de dados (*dataset*), é calculado o intervalo de tempo correspondente (primeira e última amostra). Depois disso, o intervalo de tempo é então comparado com a anotação dos eventos sonoros para se obter a classe pertencente: a classe de batida da bola será considerada se, na anotação de eventos sonoros, existir algum registo (de batida da bola) coincidente no tempo; caso contrário, será considerada a classe de ruído. A verificação de registos coincidentes no tempo obedece a uma condição pré-estabelecida, a qual define o critério de coincidência (por exemplo, a percentagem da sobreposição da janela de evento ao evento sonoro). Um exemplo será etiquetado com a classe de batida da bola, sempre que aquela condição se verificar.

3.2.5 Construção do conjunto de dados (*dataset*)

As configurações de HyP_data incluem (entre outros aspetos) o conjunto das diferentes características (*audio features*) a serem extraídas, e os intervalos de valores dos parâmetros destinados à varredura de áudio. Essas configurações são especificadas textualmente pelo utilizador e seguem um esquema formal que é (automaticamente) processado e usado para gerar os *datasets* de raiz. Portanto, a geração de cada *dataset* depende apenas do esquema de HyP_data (HyP_data_scheme), e cada configuração de valores dos HyP_data origina um *dataset* diferente.

A matriz de características é construída com os valores obtidos através do processo de extração de características do áudio (*audio feature extraction*). A implementação é feita com código genérico; a aplicação desconhece, à partida, as características que vão governar a matriz. Portanto, é necessário iterar sobre a lista de *audio features* definida no esquema de HyP_data, e para cada uma delas: aplicar o processo de extração, e concatenar o resultado (que será também uma matriz) à matriz principal. De seguida é apresentado um exemplo demonstrativo de uma matriz de características, X, considerando 'number of segments' = 5, e um conjunto de HyP_data_feature = {*feature a*,

feature b}.

$$\left[\mathrm{X}\right] = \overset{\textit{feature 'a'}}{\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & a_{n,5} \end{bmatrix}} \text{ concatenada com } \overset{\textit{feature 'b'}}{\begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{n,1} & b_{n,2} & b_{n,3} & b_{n,4} & b_{n,5} \end{bmatrix}}$$

$$<=> \left[\mathrm{X}\right] = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & a_{n,5} & b_{n,1} & b_{n,2} & b_{n,3} & b_{n,4} & b_{n,5} \end{bmatrix}$$

Uma vez que os sons impulsivos (batida de bola na raquete) ocorrem de forma esparsa quando comparados com todos os outros sons (do mesmo áudio), o *dataset* tende a ficar desequilibrado. Espera-se que o número de instâncias do *dataset* etiquetadas com a classe <batida de bola na raquete> seja muito menor do que aquelas etiquetadas como <ruído de fundo>. Para equilibrar o *dataset*, é considerada a técnica de sub-amostragem. O objetivo é reduzir o número da classe em maioria [5], neste caso, a classe do ruído de fundo.

3.2.6 Geração e avaliação do modelo

O sistema proposto explora automaticamente o espaço de HyP_learn definido, alimentado pelos *datasets* previamente gerados no seguimento da exploração do espaço de HyP_data.

Para explorar o espaço de HyP_learn, são utilizadas as técnicas grid-search e Bayesian-optimization, sendo o espaço de pesquisa definido no esquema HyP_learn. A técnica Bayesian-optimization é particularmente útil quando a busca é realizada em híper-parâmetros que têm espaços de valores contínuos, como é o exemplo do *learning-rate* das redes neuronais do tipo *Feed-forward*.

Em relação à avaliação de cada modelo, quando o mesmo procedimento de validação cruzada (*cross-validation*) e o mesmo *dataset* são usados tanto para ajustar (os HyP_learn) como para selecionar um modelo, é provável que isso leve a uma avaliação otimisticamente tendenciosa do desempenho do modelo [9]. Para evitar esse problema, e obter uma medida de desempenho e capacidade de generalização mais realista do modelo, neste trabalho é utilizada a técnica *nested cross-validation* (também conhecida por *double cross-validation*). Esta técnica combina duas camadas de validação cruzada:

- uma camada externa para avaliar o desempenho geral do modelo, e
- uma camada interna para optimizar os HyP_learn do modelo.

A camada interna é "aninhada"(*nested*) na camada externa.

A implementação desta técnica consiste em dividir o dataset em K folds para a camada externa (*outer cross-validation*), e com o conjunto de treino de cada fold (da camada externa) é executada a camada interna, na qual é feita a optimização dos HyP_learn. Esse conjunto (*set*) da camada interna é dividido noutros k folds (“inner cross-validation”) com conjunto de treino e conjunto de validação. Assim, a avaliação de cada configuração de HyP_learn não tem a oportunidade de se superajustar (*overfitting*) ao dataset original, pois está exposta apenas a um subconjunto do conjunto de dados fornecido pelo procedimento de validação cruzada externa. É utilizada a técnica *Stratified K-Fold*, de forma a preservar uma distribuição idêntica de exemplos de cada classe em cada fold, em relação ao *dataset* original.

Neste trabalho são exploradas as técnicas grid-search e Bayes-search aplicadas na camada interna da *cross-validation*. Na camada externa é avaliado, com o conjunto de teste, o desempenho geral do modelo otimizado (pela camada interna). No final, o desempenho apresentado será a média dos resultados dos K folds da camada externa.

Na Figura 3.7 está representada uma ilustração demonstrativa da técnica *nested-cross validation*, com uma configuração de 5 folds na camada externa e 3 folds na camada interna. É também possível visualizar, na Figura 3.7, o conjunto de dados de validação na camada interna (associado à avaliação de híper-parâmetros na fase de treino), e o conjunto de dados de teste na camada externa (associado à avaliação do modelo).

Pré-processamento do *dataset*. Antes do *dataset* ser processado pelo algoritmo de classificação, é usada uma técnica de pré-processamento para uniformizar a escala das *features*, removendo a média e dimensionando o valor das *features* de acordo com a variância. Entende-se que não há necessidade de outros passos de pré-processamento, pois os *datasets* são gerados pelo sistema (i.e., não são externos), logo estão controladas possíveis situações tais como: colunas de valor não-numérico[26], valores em falta, *features* indesejadas, etc. Para automatizar o passo de pré-processamento seguido do classificador, foi criado um Pipeline[27].

[26]O processo de conversão de valores alfanuméricos em números, é chamado *Label Encoding*. Existe também o processo chamado *One-Hot Encoding*, no qual uma coluna pode ser dividida em múltiplas colunas, sendo o número de novas colunas igual ao número de valores alfanuméricos exclusivos da coluna original [2].

[27]O Pipeline permite aplicar sequencialmente uma lista de transformadores para pré-processar os dados e, opcionalmente, concluir a sequência com um classificador. Referência: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

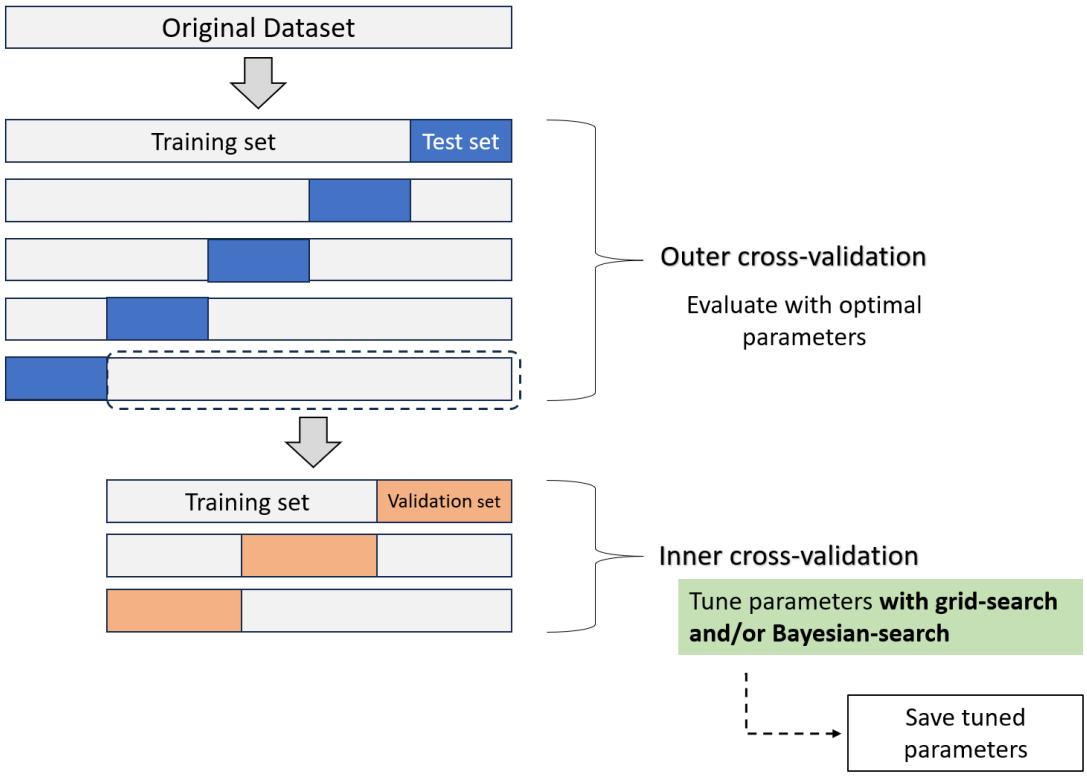

Figura 3.7: Técnica *nested cross-validation*, com uma configuração de 5x3.

No final de todo o processo, são apresentados os resultados do desempenho de todos os modelos, assim como, os melhores híper-parâmetros de cada modelo.

4

Implementação

No desenvolvimento do código que implementa o sistema, foi definida uma estrutura em 2 grandes "blocos" principais, respeitando as duas componentes distintas do sistema, conforme mostrado na Figura 3.4. As duas componentes dizem-se distintas pois são funcionalmente independentes, embora exista uma dependência temporal, onde a componente que representa a 2ª fase do sistema (processamento dos *datasets*) depende dos *datasets* já estarem criados. Assim, o bloco que implementa a 1ª fase deve ser executado primeiramente, e só após concluída a sua execução, é que a 2ª fase pode ser executada. O sistema é implementado usando a linguagem de programação Python.

A interação do utilizador com a aplicação, em ambas as fases, é feita através dos esquemas de HyP_data e HyP_learn. Os esquemas são guardados numa estrutura de dados persistente, em concreto, um ficheiro JSON.

Neste capítulo é justificada a escolha do ecossistema Python, e são apresentados os aspetos importantes da implementação, considerando a distinção das duas fases.

4.1 Tecnologia utilizada

A escolha da linguagem de programação Python foi feita considerando o vasto ecossistema de bibliotecas e frameworks desenhadas especialmente para o desenvolvimento de modelos de aprendizagem automática (e.g. TensorFlow; Keras[1]; Scikit-Learn); as

[1]Keras website, https://keras.io/

bibliotecas disponíveis para o processamento de sinais de áudio (e.g., Librosa[2]); e as bibliotecas para manipulação e processamento de dados (e.g. NumPy; Pandas). O Scikit-learn é uma das bibliotecas de código aberto mais utilizadas para aprendizagem automática [2].

4.2 Primeira fase - geração dos *datasets*

A metodologia proposta para o esquema de HyP_data permite que os *datasets* sejam gerados automaticamente, com código genérico, e sem a necessidade de intervenção do utilizador. O esquema consiste na especificação do espaço de pesquisa de HyP_data_global, e de HyP_data_feature. Para cada *audio feature*, existe a respetiva função responsável por obter o seu valor num excerto de áudio. A função retorna a matriz correspondente ao varrimento sobre o áudio, onde cada linha corresponde a uma janela de evento, e cada coluna corresponde ao valor de uma *feature* num segmento da janela. Para o processamento do sinal de áudio é utilizada a biblioteca Librosa. Essa função é armazenada num módulo separado, ao qual a aplicação acede e obtém a função através do seu nome (o qual é especificado em HyP_data_feature). Assim, possibilita-se a extensão do sistema com incorporação de funções adicionais (modelo *plug-in*) para explorar outras *audio features*.

O programa acede ao ficheiro HyP_data e carrega o conteúdo, de forma a ter uma representação[3] do esquema. A partir dessa representação são criadas N cópias, onde N corresponde ao número de combinações de valores de HyP_data. Cada cópia terá valores concretos (i.e., uma combinação) dos HyP_data e representará uma configuração. Essas cópias são também guardadas em estrutura de dados persistente, para que posteriormente seja possível conhecer os HyP_data relativos a cada *dataset* gerado.

O processo de construção de cada *dataset* implementa a metodologia descrita no capítulo 3.2. São criados N *datasets*, a partir das N configurações de HyP_data, e são guardados em ficheiro CSV[4] numa directoria cujo caminho é registado no ficheiro de configuração do sistema.

No processo de etiquetagem automática dos dados (*automatic class labeling*), é predefinida a condição (critério de coincidência no tempo) indicada na Listagem 4.1.

[2]Librosa website, https://librosa.org/

[3]Considerando a implementação na linguagem Python, o ficheiro HyP_data.json é carregado para um dicionário.

[4]Ficheiro *Comma Separated Value* (CSV). A informação adicional sobre este formato foi consultada em https://creativyst.com/Doc/Articles/CSV/CSV01.shtml.

```python
# init_idx      - primeira amostra do exemplo (do conjunto de dados)
# final_idx     - última amostra do exemplo (do conjunto de dados)

# first_sample  - primeira amosta da anotação de eventos sonoros
# last_sample   - última amosta da anotação de eventos sonoros

condition = final_idx >= first_sample and init_idx <= last_sample
if condition: ball_hit = True
```

Listagem 4.1: Condição para a etiquetagem automática dos dados. (Código: Python)

4.3 Segunda fase - processamento dos *datasets*

Na definição dos classificadores que vão processar os *datasets*, é seguida a mesma metodologia das funções de "audio feature extraction"implementada na 1ª fase (geração dos *datasets*). Assim, para cada classificador definido no esquema de HyP_learn, deve existir a respetiva função responsável por "entregar" o Pipeline contendo o pré-processamento dos dados seguido do classificador (*estimator*). Com esta metodologia, possibilita-se a extensão do sistema com incorporação de funções adicionais (modelo *plug-in*) para explorar outros classificadores.

O utilizador define, no esquema de HyP_learn: os classificadores a considerar, e para cada classificador, o esquema dos híper-parâmetros que se pretender otimizar.

Cada classificador representa uma configuração de HyP_learn. Cada configuração é testada com todos os *datasets* gerados na 1ª fase. Por exemplo: tendo-se 10 *datasets* e 2 classificadores (e.g., MLP e SVM), serão testados 20 modelos.

As técnicas grid-search[5] e Bayes-search[6] são aplicadas na camada interna da *cross-validation*, com o intervalo de híper-parâmetros de cada modelo que está definido no esquema de HyP_learn.

A métrica de avaliação do modelo é um parâmetro configurável do sistema, tal como a configuração de *folds* da técnica *nested cross-validation*.

O processo de geração e avaliação dos modelos implementa a metodologia descrita no capítulo 3.2.

[5] A implementação da técnica grid-search recorre à biblioteca sklearn.model_selection.GridSearchCV. Website: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[6] A implementação da técnica Bayes-search recorre à biblioteca scikit-optimize.BayesSearchCV. Website: https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html

4.4 Retorno do sistema – apresentação de resultados

Os resultados a apresentar incluem o desempenho de todos os modelos, assim como, os melhores híper-parâmetros de cada modelo, incluindo o nome do *dataset*. O desempenho é avaliado de acordo com a métrica definida.

O sistema permite assim, avaliar o impacte individual de cada configuração de HyP_data, e HyP_learn, no desempenho global do modelo.

5

Resultados experimentais

O objetivo da experiência realizada, apresentada neste capítulo, é o suporte experimental da hipótese de trabalho.

5.1 Configuração experimental

Para avaliar o impacte da variação do *dataset* no desempenho global do modelo, são testados diferentes valores dos parâmetros do varrimento sobre o áudio para a extração de características. Assim, na configuração experimental, é considerado o seguinte conjunto de HyP_data:

- HyP_data_global = {
  - Event_length: 0.5 segundos,
  - Sampling_frequency: 44100 Hz,
  - Number_of_shifted_samples: [4096, 2048, 1024] ,
  - Number_of_segments: ajustado de acordo com a variação do 'number of shifted samples' }
- HyP_data_feature = {
  - Onset Detect,

- Root Mean Square,
- Band Energy Ratio,
- Spectral flux }

Nesta experiência foi aproveitado o contributo do trabalho relacionado [12], em particular, os vídeos amadores de jogos de padel e a anotação manual de eventos. Nesse trabalho, duração dos eventos ('event length') foi pré-definida para meio segundo. Portanto, sendo todos aqueles vídeos recolhidos no mesmo tipo de ambiente (*indoor*), então nesta experiência é também considerado o valor de meio segundo (0.5 s) para o HyP_data_global 'event length'.

Para o HyP_data_global 'sampling frequency' (frequência de amostragem), nesta experiência é definido o valor 44100 [Hz], por ser um dos valores padrão utilizados [1].

É testada a variação do número de amostras que a janela desliza a cada iteração ('number of shifted samples'), com três valores distintos. O 'number of segments' é ajustado de acordo com a variação do 'number of shifted samples', embora esse ajuste não seja obrigatório, pois os conceitos são independentes.

A escolha dos HyP_data_feature foi baseada nos fundamentos apresentados no capítulo 3.1. Foram também considerados os resultados apresentados no trabalho relacionado [12], nos quais a utilização das *audio features* Onset Detect, Root Mean Square e Spectral Flux produziu um resultado aceitável. Adicionalmente àquelas três *audio features*, foi considerada também a Band Energy Ratio, para complementar a representatividade de características do domínio da frequência no *dataset*.

Com as 3 configurações de HyP_data, foram gerados 3 datasets, descritos doravante como 'Dataset_1', 'Dataset_2', e 'Dataset_3'. Na Tabela 5.2 estão representadas as configurações de HyP_data, incluindo o *dataset* gerado a partir de cada configuração. Nessa tabela está omisso o conjunto de HyP_data_feature, contudo ele é igual em todas as configurações.

Como fonte de dados para a geração dos *datasets*, nesta experiência, foram utilizados 30 vídeos amadores de jogos de padel, em ambiente *indoor*, e com pouco ruído (sem público a assistir; sem música). Os vídeos têm a duração de aproximadamente 1 minuto (cada), totalizando o tempo de gravação de 29 minutos e 31 segundos. Na Tabela 5.1 é apresentada a informação sobre cada "dataset".

Relativamente aos HyP_learn, são considerados dois algoritmos para a classificação: rede neuronal *Multi Layer Perceptron (MLP)*, e *Support Vector Machine (SVM)*. A escolha dos algoritmos (MLP e SVM) para a experiência, deve-se ao facto de eles terem apresentado bons resultados em trabalhos relacionados [3, 4, 7, 8, 12, 15, 24, 28].

Tabela 5.1: *Datasets* gerados

Dataset	Dimensão (linhas)	Dimensão (colunas)	Exemplos negativos	Exemplos positivos
Dataset_1	7852	20	4017	3835
Dataset_2	15869	44	7959	7910
Dataset_3	31740	88	15924	15816

Consideraram-se alguns dos HyP_learn pertencentes àqueles algoritmos, e definiu-se um espaço de pesquisa para cada um deles.

Para a SVM, foi considerado o híper-parâmetro C para otimização. A configuração dos híper-parâmetros do classificador foi definida da seguinte forma:

- *Kernel* – RBF.
- C – espaço de pesquisa (1, 40).
- Gamma – *'scale'*.

Para a rede MLP, foram considerados dois híper-parâmetros para otimização: taxa de aprendizagem inicial, e máximo número de iterações. A configuração dos híper-parâmetros do classificador foi definida da seguinte forma:

- Número de camadas – 1 camada escondida.
- Número de neurónios por camada – 100.
- Tamanho do *batch* – ajustado pelo algoritmo, *batch_size=min(200, n_samples)*.
- Função de activação – ReLU (camada escondida).
- Algoritmo de otimização – *Adam*. (*epsilon* =1e-8).
- Taxa de aprendizagem inicial – espaço de pesquisa (0.0001, 0.01).
- Máximo número de iterações (número de épocas) – espaço de pesquisa (1, 100). O algoritmo de otimização (*Adam*) itera até atingir a convergência (determinada pela tolerância = 1e-4) ou este número de iterações.
- Regularização – L2, termo de regularização =0.0001. O termo de regularização L2 é dividido pelo tamanho do exemplo quando somado à perda.
- *Early stopping* – desligado.

Tabela 5.2: Configurações de HyP_data.

Dataset	Event length (seconds)	Number of segments	Bandwidth (Hz)	**Number of shifted samples**
Dataset_1	0.5	5	44100	**4096**
Dataset_2	0.5	11	44100	**2048**
Dataset_3	0.5	22	44100	**1024**

Tabela 5.3: Espaço de pesquisa de HyP_learn.

Model	HyP_learn	Range of values (start, stop, step (if applicable))
MLP	learning rate	0.0001, 0.01, step = 0.0005 for the grid-search
MLP	max iterations	1, 100, step = 5 for the grid-search
SVM	C	1, 40, step = 3 for the grid-search

- *Shuffle* – ligado. A ordem dos exemplos é baralhada a cada iteração.

Na Tabela 5.3 é apresentado o espaço de pesquisa de HyP_learn.

Para a técnica *Nested cross-validation* foi definida uma configuração com 10 folds na camada externa e 3 folds na camada interna.

O desempenho dos modelos foi avaliado através da taxa de sucesso (*accuracy*). Uma vez que os três *datasets* estão equilibrados, e é utilizada a técnica *Stratified K-Fold*, então é possível "confiar" no resultado da avaliação com esta métrica. Contudo, e conforme abordado no capítulo 3.1.7 (Avaliação do modelo), devem ser consideradas outras métricas, para avaliar os modelos de forma mais rigorosa. Em próximas experiências deverá ser considerara, pelo menos, a métrica *F1 Score*, pois é útil em problemas onde o *dataset* é desequilibrado.

Na Tabela 5.4 são mostrados os resultados da experiência. Foram usadas as técnicas *grid-search* e *Bayesian-optimization*.

5.2 Análise dos resultados

Começando pelos algoritmos de classificação testados – MLP e SVM – o resultado mostra um desempenho equivalente de ambos, com uma vantagem muito ligeira para a rede neuronal MLP. A taxa de sucesso (*accuracy*) atingiu o valor de 93%, contudo, isso pode dever-se à fraca variabilidade da fonte de dados.

Tabela 5.4: Apresentação dos resultados experimentais. (Legenda: (GS) - grid-search ; (BO) - Bayesian search)

Dataset	Model	**Score (Accuracy)**	Best HyP_learn	
Dataset_3	MLP	**0.930653**	'learning rate': 0.000600, 'max iter': 76	(GS)
Dataset_3	MLP	**0.929235**	'learning rate': 0.001036, 'max iter': 67	(BO)
Dataset_3	SVM	**0.928574**	'C': 31	(BO)
Dataset_3	SVM	**0.928574**	'C': 31	(GS)
Dataset_2	MLP	**0.921098**	'learning rate': 0.001360, 'max iter': 44	(BO)
Dataset_2	MLP	**0.920090**	'learning rate': 0.0016, 'max iter': 36	(GS)
Dataset_2	SVM	**0.919018**	'C': 19	(GS)
Dataset_2	SVM	**0.918640**	'C': 23	(BO)
Dataset_1	MLP	**0.900521**	'learning rate': 0.0041, 'max iter': 86	(GS)
Dataset_1	MLP	**0.899885**	'learning rate': 0.002799, 'max iter': 78	(BO)
Dataset_1	SVM	**0.894536**	'C': 36	(BO)
Dataset_1	SVM	**0.894408**	'C': 28	(GS)

No que diz respeito às técnicas de pesquisa de híper-parâmetros, *grid-search* e *Bayesian-optimization*, verificam-se as suas vantagens e desvantagens já documentadas na literatura [2, 13]. A diferença mais significativa observada entre as duas técnicas foi o custo computacional: utilizando um computador pessoal (equipado com um processador AMD Ryzen 7 3700U e 12GB de memória RAM), a experiência realizada com o conjunto de dados 'Dataset_3' demorou mais do dobro do tempo (cerca de 26 horas com a técnica *grid-search*, e cerca de 12 horas com a técnica *Bayesian-optimization*).

Observando os *datasets* gerados, verifica-se que o varrimento sobre o áudio com um deslizamento menor da janela, resulta num *dataset* com mais exemplos, e num melhor desempenho global do modelo. Este desempenho pode justificar-se não apenas pela (maior) dimensão do *dataset*, mas também pela consequência do deslizamento da janela, que resulta numa maior probabilidade de ser captada a duração total do evento.

Os resultados permitem validar a técnica do varrimento sobre o áudio para extração de características, pois a variação do conjunto HyP_data_global teve influência no desempenho global do sistema.

A variação de HyP_data permite gerar *datasets* com diferentes dimensões, com o mesmo conjunto de vídeos.

6

Conclusões e Trabalho futuro

6.1 Conclusões

O sistema desenvolvido demonstra o impacte significativo da exploração do espaço de híper-parâmetros na otimização do desempenho de todo o processo de classificação automática de sons impulsivos. Demonstra também, a importância não apenas dos HyP_learn, mas também dos HyP_data.

Os valores de HyP encontrados pelo sistema permitiram ao modelo atingir uma taxa de sucesso (*accuracy*) de 93%. No entanto, essa taxa de sucesso pode ser mais baixa se o sistema for testado com vídeos gravados em diferentes condições.

A variação de HyP_data permite que, com o mesmo conjunto de vídeos, sejam gerados *datasets* de diferentes dimensões, tendo grande influência no desempenho global do sistema.

Em relação às técnicas de pesquisa de híper-parâmetros, a *Bayesian-optimization* mostra ser a técnica mais adequada para este problema, uma vez que se lida com uma grande dimensão do espaço de configurações.

A metodologia proposta permite que todo o processo seja automatizado, desde a geração dos *datasets* a processar, passando pela aprendizagem de modelos de classificação, e terminando com a avaliação dos modelos aprendidos. Com base nos resultados alcançados, podemos concluir que a metodologia demonstrou ser capaz de otimizar tanto os HyP_data como os HyP_learn.

Verificou-se que as características do som, e os parâmetros que definem o seu processamento, têm uma grande influência no *dataset* que é construído. O *dataset*, por sua vez, tem influência significativa no desempenho global do sistema. A hipótese de trabalho foi validada.

6.2 Trabalho futuro

Com os testes realizados, a hipótese de trabalho foi validada, mas para estar consistente, poderão ser feitos mais testes, nomeadamente, a exploração de mais híper-parâmetros (HyP_data e HyP_learn).

Capacitar o sistema para suportar HyP de métodos de análise baseados em imagens, tais como, Redes Neuronais Convolucionais (CNNs). Isto envolverá, à partida, um novo método de geração de datasets, os quais serão compostos por representações do som em imagens, mais conctretamente, o espectrograma. Este melhoramento poderá ter, como vantagem, o aumento do desempenho do sistema, fruto das conhecidas vantagens das CNNs.

Estudar a possibilidade de uma metodologia que utilize simultaneamente as técnicas grid-search e Bayesian-optimization, aproveitando as vantagens de cada uma. Sugestão de metodologia: a técnica grid-search ser usada para explorar o espaço de híper-parâmetros com valores discretos, e a técnica Bayesian-optimization ser usada nos híper-parâmetros com valores contínuos. Considerando uma rede neuronal (MLP), ela tem híper-parâmetros com espaço de valores contínuos, e também tem híper-parâmetros com espaço de valores discretos.

Aumentar a quantidade e variabilidade do conjunto de dados que serve de base para a criação dos *datasets*. Verificou-se que é importante ter dados recolhidos em diferentes condições e ambientes.

Desenvolvimento de uma aplicação *web* (interface gráfica) para dar suporte à utilização do sistema. A aplicação poderá também abranger as seguintes funcionalidades: a) melhoria contínua dos classificadores - via extensão dos *datasets* com incorporação dos dados de *feedback* (válido) fornecido por utilizadores, e b) evolução contínua de todo o processo - via extensão do sistema com incorporação de funções adicionais (modelo *plug-in*) para explorar diferentes híper-parâmetros (e.g., outras *audio features*).

Referências

[1] Adobe. "Sample rates and audio sampling: a guide for beginners." (2024), URL: https://www.adobe.com/uk/creativecloud/video/discover/audio-sampling.html.

[2] Tanay Agrawal, *Hyperparameter Optimization in Machine Learning*. Apress, 2021, ISBN: 978-1-4842-6579-6.

[3] Momin Uppal Ahmed Talal & Abubakr Muhammad, "Improving efficiency and reliability of gunshot detection systems", *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013. URL: https://doi.org/10.1109/ICASSP.2013.6637700.

[4] Yüksel Arslan, "Impulsive Sound Detection by a Novel Energy Formula and its Usage for Gunshot Recognition", *Cornell University, arxiv.org*, 2017. URL: https://doi.org/10.48550/arXiv.1706.08759.

[5] Will Badr, "Having an imbalanced dataset? Here is how you can fix it.", *Towards Data Science*, 2019. URL: https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb.

[6] David Bawiec. "What's the Real Difference Between .wav, .aiff, .mp3, and .m4a?" (2019), URL: https://www.izotope.com/en/learn/whats-the-difference-between-file-formats.html.

[7] James Bergstra & Yoshua Bengio, "Random Search for Hyper-Parameter Optimization", *Journal of Machine Learning Research*, 2012. URL: https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf?ref=blog.floydhub.com.

[8] Martí Bolet Boixeda & Elisa Sayrol, "Urban Sounds Classification using Deep Learning, final course project, Degree in Telecommunications Engineering", Trabalho Final de Curso, Universitat Politècnica de Catalunya, Barcelona, Spain, 2019.

[9] Jason Brownlee. "Nested Cross-Validation for Machine Learning with Python. Machine Learning Mastery". (nov. de 2021), URL: https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python/.

[10] Devopedia. "Audio Feature Extraction". (mai. de 2021), URL: https://devopedia.org/audio-feature-extraction.

[11] Mohamed Maher Elshawi Radwa & Sherif Sakr, "Automated Machine Learning: State-of-The-Art and Open Challenges", *Cornell University, arxiv.org*, 2019. URL: https://doi.org/10.48550/arXiv.1906.02287.

[12] Carina Fernandes Paulo Trigo Joel Paulo & Paulo Vieira, "Anotação de Eventos Sonoros em Vídeo", Trabalho Final de Curso, Instituto Superior de Engenharia de Lisboa (ISEL), Lisbon, Portugal, 2022.

[13] Matthias Feurer & Frank Hutter, *Automated Machine Learning. Methods, Systems, Challenges*. Springer, 2019, ISBN: 978-3-030-05317-8.

[14] Ellis DP Gemmeke JF et al. S. Hershey S. Chaudhuri, "CNN Architectures for Large-Scale Audio Classification", *Cornell University, arxiv.org*, 2017. URL: https://doi.org/10.48550/arXiv.1609.09430.

[15] Martin Hrabina & Milan Sigmund, "Gunshot recognition using low level features in the time domain", *2018 28th International Conference Radioelektronika (RADIOELEKTRONIKA)*, 2018. URL: https://doi.org/10.1109/RADIOELEK.2018.8376372.

[16] Uday Kiran. "MFCC Technique for Speech Recognition". (2023), URL: https://www.analyticsvidhya.com/blog/2021/06/mfcc-technique-for-speech-recognition/.

[17] Giulia DeSalvo Afshin Rostamizadeh Lisha Li Kevin Jamieson & Ameet Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization", *Journal of Machine Learning Research*, 2018. URL: https://doi.org/10.48550/arXiv.1603.06560.

[18] machinelearning.wtf. "Harmonic Precision-Recall Mean (F1 Score)". (2017), URL: https://machinelearning.wtf/terms/harmonic-precision-recall-mean-f1-score/.

[19] MAPLE Lab. "Amplitude Envelope". (2024), URL: https://maplelab.net/overview/amplitude-envelope/.

[20] Nathan Van der Rest Sam McGuire, *The Musical Art of Synthesis*. Focal Press, 2016, ISBN: 978-1-315-73759-1 (ebk).

[21] Meyda. "Audio feature extraction for JavaScript." (2022), URL: https://meyda.js.org/audio-features.html.

[22] Tom M. Mitchell, *Machine Learning*. McGraw Hill, 1997, ISBN: 0070428077.

[23] Macarena Varela Nalla Ravali & Marc Oispuu, "Evaluation of Image Classification Networks on Impulse Sound Classification Task", *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2021. URL: https://doi.org/10.1109/MFI52462.2021.9591202.

[24] H. Kadri Rabaoui A. & N. Ellouze, "New approaches based on One-Class SVMs for impulsive sounds recognition tasks", *2008 IEEE Workshop on Machine Learning for Signal Processing*, 2008. URL: https://doi.org/10.1109/MLSP.2008.4685494.

[25] Drishti Sharma. "Analysis of Zero Crossing Rates of Different Music Genre Tracks". (2022), URL: https://www.analyticsvidhya.com/blog/2022/01/analysis-of-zero-crossing-rates-of-different-music-genre-tracks/.

[26] Zhandos; Omarov Nurzhan; Sadykova Bibigul; Zhekambayeva Maigul; et al. Smailov Nurzhigit; Dosbayev, "A Novel Deep CNN-RNN Approach for Real-time Impulsive Sound Detection to Detect Dangerous Events", *International Journal of Advanced Computer Science and Applications, West Yorkshire*, vol. 14, n.º 4, 2023.

[27] Hugo Larochelle Jasper Snoek & Ryan P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms", *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

[28] Batyrkhan Omarov Suliman Azizah & Zhandos Dosbayev, "Detection of impulsive sounds in stream of audio signals", *2020 8th International Conference on Information Technology and Multimedia (ICIMU)*, 2020. URL: https://doi.org/10.1109/ICIMU49871.2020.9243540.

[29] Olivia Tanuwidjaja. "Get To Know Audio Feature Extraction in Python". (2022), URL: https://towardsdatascience.com/get-to-know-audio-feature-extraction-in-python-a499fdaefe42.

[30] David Lo Tardif Bruno & Rafik Goubran, "Gunshot Sound Measurement and Analysis", *2021 IEEE Sensors Applications Symposium (SAS)*, 2021. URL: https://doi.org/10.1109/SAS51076.2021.9530145.

[31] Vinícius Rodrigues. "Entenda o que é AUC e ROC nos modelos de Machine Learning". (2018), URL: https://medium.com/bio-data-blog/entenda-o-que-%C3%A9-auc-e-roc-nos-modelos-de-machine-learning-8191fb4df772.

Repositório do projeto de implementação do sistema

O projeto de implementação do sistema está disponível no seguinte repositório Git: https://github.com/asmendes1/TFM-30569-Projeto.git

Na diretoria "raiz" encontra-se o ficheiro "Readme" com as instruções de configuração.