U.PORTO

MESTRADO
MULTIMÉDIA - ESPECIALIZAÇÃO EM MÚSICA INTERATIVA E DESIGN DE SOM

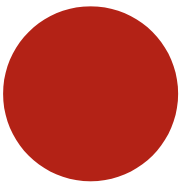
HARMONIZANDO EMOÇÕES: UMA ABORDAGEM MULTIMODAL PARA RECONHECIMENTO EMOCIONAL NA MÚSICA

Ricardo Martins Albuquerque

M
2024

FACULDADES PARTICIPANTES:

FACULDADE DE ENGENHARIA
FACULDADE DE BELAS ARTES
FACULDADE DE CIÊNCIAS
FACULDADE DE ECONOMIA
FACULDADE DE LETRAS



Harmonizando Emoções: Uma Abordagem Multimodal para Reconhecimento Emocional na Música

Ricardo Martins Albuquerque

Mestrado em Multimédia da Universidade do Porto

Aprovado em provas públicas pelo Júri:

Presidente: Nome do Presidente (Título)

Vogal Externo: Nome do Arguente (Título)

Orientador: João Manuel R. S. Tavares (Professor Catedrático)

Resumo

A música é um meio poderoso capaz de transmitir uma grande variedade de emoções. Nos últimos anos, o campo de Reconhecimento de Emoções Musicais (*Music Emotion Recognition* - MER) tem atraído uma atenção significativa tanto no ramo académico como industrial pelas suas aplicações em domínios como sistemas de recomendação, extração musical, composição automática de música e ainda psicoterapia. Esta Dissertação explora a integração de abordagens multimodais para o reconhecimento de emoções, centrando-se na utilização de técnicas de *Machine Learning*, *Deep Learning* e *ensemble learning* para o reconhecimento de emoções.

A metodologia desta investigação envolveu uma revisão da literatura, a análise de diversos conjuntos de dados, a implementação de técnicas MER e o desenvolvimento de uma estrutura multimodal para o reconhecimento emocional. Esta estrutura foi validada através de múltiplas experiências e de uma análise comparativa com abordagens atuais na literatura.

Esta Dissertação começa com uma revisão do estado da arte, apresentando trabalhos relacionados no ramo de MER, técnicas de extração de características e abordagens existentes. Depois, é feita uma apresentação detalhada do problema que se pretende resolver, seguida da descrição da solução proposta e dos aspetos metodológicos relevantes. A implementação da abordagem é descrita na sua totalidade, detalhando todas as arquiteturas utilizadas.

A investigação realizada demonstrou que os modelos de *Deep Learning*, especialmente aqueles com mecanismos de atenção, ultrapassam significativamente as abordagens *de Machine Learning* no reconhecimento emocional. Além disso, a implementação de modelos de fusão e a incorporação de características áudio e lírica resultaram em melhorias substanciais no desempenho dos modelos.

Contudo, é crucial reconhecer as limitações do estudo. A subjetividade inerente à perceção emocional, as potenciais lacunas nos conjuntos de dados utilizados e a dificuldade em interpretar os resultados obtidos são desafios importantes. Estas limitações salientam a necessidade de investigação contínua e abordagens mais transparentes no campo do reconhecimento emocional em música, para melhorar a compreensão e a aplicabilidade destes modelos.

Abstract

Music is a powerful medium capable of conveying a wide range of emotions. In recent years, the field of Music Emotion Recognition (MER) has garnered significant attention from both academia and industry due to its applications in domains such as recommendation systems, music retrieval, automatic music composition, and psychotherapy. This dissertation explores the integration of multimodal approaches for emotion recognition in music, focusing on the extraction of musical features and the use of Machine Learning, Deep Learning and ensemble learning techniques for emotional recognition.

The methodology of this research involves a literature review, analysis of multiple datasets, implementation and evaluation of MER techniques, and the development of a multimodal framework. This framework is validated through various experiments and a comparative analysis with current approaches in the literature.

The Dissertation begins with a comprehensive review of the state-of-the-art in MER, presenting related work, feature extraction techniques, and existing MER approaches. It then delves into the detailed presentation of the problem this dissertation aims to solve, followed by a description of the proposed solution and relevant methodological aspects. The implementation of the approach is described in its entirety, detailing all the employed architectures.

The conducted research demonstrated that Deep Learning models, especially those with attention mechanisms, significantly outperform traditional Machine Learning approaches in emotional recognition. Furthermore, the implementation of fusion models and the incorporation of audio and lyrical features resulted in substantial improvements in model performance.

However, it is crucial to acknowledge the study's limitations. The inherent subjectivity of emotional perception, potential gaps in the datasets used, and the difficulty in interpreting the obtained results are significant challenges. These limitations highlight the need for continued research and more transparent approaches in the field of emotional recognition in music, to improve the understanding and applicability of these models.

Agradecimentos

Aos meus pais, aos amigos que foram, aos amigos que ficaram, à Adriana, à minha terapeuta, à Bibinha e ao professor Augusto.

Ricardo

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas

MER	*Music Emotion Recognition*
MIR	*Music Information Retrieval*
NLP	*Natural Language Processing*
EEG	Electroencefalograma
AMC	*Audio Mood Recognition*
ML	*Machine Learning*
DL	*Deep Learning*
WASABI	*Web Audio Semantic Aggregated in the Browser for Indexation*
MIDI	*Musical Instrument Digital Interface*
BOW	*Bag-of-Words*
POS	*Part-of-Speech*
BERT	*Bidirecional Encoder Representations from Transformers*
fMRI	Ressonância Magnética Funcional
CNN	*Convolutional Neural Networks*
RNN	*Recurrent Neural Networks*
Bi-RNN	*Bi-Directional Recurrent Neural Networks*
LSTM	*Long Short-Term Memory*
Bi-LSTM	*Bi-Directional Long Short-Term Memory*
DBLSTM	*Deep Bidirectional Long Short-Term Memory*
BCRSN	*Bi-directional Convolutional Recurrent Sparse Network*
CQT	*Constant-Q Transform*
MCA	*Multi-Scale Context-Based Attention Model*
MLR	*Multinomial Logistic Regression*
SVR	*Support Vector Regression*
SVM	*Support Vector Machine*

BPM	*Beats per Minute*
HAN	*Hierarchical Attention Networks*
GRU	*Gated Recurrent Unit*
SSM	*Self-similarity matrix*
LDA	*Latent Dirichlet Allocation*
MAE	*Mean Absolute Error*
MSE	*Mean Squared Error*
CMA	*Crossmodal Attention*
MFCC	*Mel-frequency cepstral coefficients*

1. Introdução

A música é um meio de comunicação poderoso com a capacidade de evocar e transmitir uma vasta gama de emoções. Nos últimos anos, o campo de Reconhecimento de Emoções Musicais (*Music Emotion Recognition* - MER) tem atraído muita atenção tanto na comunidade académica como industrial (Juslin & Laukka, 2004) devido às suas aplicações em domínios de elevado potencial como sistemas de recomendação (Cheng et al. 2017), extração musical (Shen et al. 2017), composição automática de música (Gunawan et al., 2020) e psicoterapia (De Witte et al., 2020). Esta Dissertação explora a integração de abordagens multimodais para o reconhecimento de emoções na música, centrando-se na extração de características musicais e na utilização de técnicas de *Machine Learning* e *Deep Learning* e *ensemble learning* para o reconhecimento emocional.

Os principais objetivos desta Dissertação são a investigação do estado atual do ramo de MER e a exploração de múltiplas arquiteturas de *Machine Learning*, *Deep Learning* e *ensemble learning* para extração de características e reconhecimento emocional. A metodologia desta investigação envolve uma revisão da literatura, a análise de múltiplos conjuntos de dados, a implementação e avaliação de técnicas MER e o desenvolvimento de uma estrutura multimodal. Esta estrutura é validada através de múltiplas experiências e de uma análise comparativa com abordagens atuais na literatura.

1.1 Enquadramento

O ramo de MER trata-se de uma sub-tarefa no domínio da Recolha de Informação Musical (*Music Information Retrieval* – MIR) e existe na interseção de diversas disciplinas como a psicologia musical, o processamento de sinais de áudio e o processamento de linguagens naturais (*Natural Language Processing* - NLP). Este ramo de investigação envolve a utilização de métodos computacionais que permitem a extração e análise das características da música, sendo que estes estabelecem relações entre as características extraídas e os espaços emocionais, identificando no final as emoções expressas na música (Yang *et al.* 2017). Normalmente, estas características são obtidas de múltiplas fontes, como sinais de áudio, partituras, letras da música, ou até através de indicadores biológicos, com o auxílio de um eletroencefalograma (EEG) (He *et al*., 2020).

Os serviços de *streaming* de música, como *Spotify*, *Tidal* ou *Apple Music*, utilizam várias ferramentas de inteligência artificial para analisar os perfis e os hábitos musicais dos seus utilizadores, aplicando técnicas como filtragem colaborativa ou filtragem baseada no conteúdo, que permitem fornecer recomendações musicais personalizadas aos utilizadores (Schedl *et al.*, 2018). Os fatores que apresentam um maior peso nestes processos de seleção são o género musical, a popularidade da música e o contexto no qual a música é consumida. No entanto, existem outros critérios que estes serviços de *streaming* podem explorar e aplicar nesta filtragem, mas que até ao momento não apresentaram o mesmo destaque. Tal é o caso das emoções expressas na música.

Apesar dos grandes avanços no ramo de MER, ainda existem desafios e limitações em termos da qualidade dos dados disponibilizados e da subjetividade das anotações emocionais. Esta Dissertação teve como objetivo principal abordar estes desafios e explorar abordagens multimodais que tiram partido de múltiplas fontes de informação para interpretação emocional.

1.2 Problemas e Objetivos de Investigação

As principais questões abordadas nesta Dissertação foram:

- Como podemos extrair e representar de uma forma eficaz as características relevantes para o reconhecimento emocional na música?
- A integração de múltiplas modalidades, como áudio e letras, pode de facto melhorar a fiabilidade das arquiteturas MER?
- Quais as vantagens e desvantagens entre o uso das diferentes abordagens de *Machine Learning*, *Deep Learning* e *ensemble learning*, quais as características

mais pertinentes a cada uma destas propostas e qual o desempenho destas comparativamente ao contexto atual do ramo de MER?

Os objetivos desta Dissertação foram os seguintes:

- Revisão da literatura sobre as técnicas existentes no campo de MER.
- Investigação e comparação dos diversos métodos de aprendizagem automática para características áudio e letras.
- Desenvolvimento de uma estrutura MER multimodal que integra os benefícios das arquiteturas de *Machine Learning* e *Deep Learning*.
- Comparação do desempenho da metodologia proposta com abordagens relevantes na literatura.

1.3 Metodologia de Investigação

Esta Dissertação empregou uma metodologia de investigação quantitativa, recorrendo a métodos computacionais e a avaliações empíricas para responder às questões apresentadas. Assim, a metodologia de investigação envolveu os seguintes passos:

- Revisão da literatura: Levantamento de técnicas de MER existentes, assim como dos conjuntos de dados e métricas de avaliação empregues, de modo a compreender o estado atual da área e as lacunas que apresenta.
- Análise de conjuntos de dados: Avaliação e comparação dos *datasets* disponíveis, tendo em conta a sua dimensão, anotações emocionais e modalidades disponibilizadas.
- Desenvolvimento dos modelos: Criação de diversos modelos *Machine Learning* e *Deep Learning*, sendo os resultados dos modelos com melhor desempenho de cada modalidade utilizados na criação de um modelo de fusão.
- Avaliação experimental: Avaliação da estrutura proposta através de diversas métricas de avaliação. Comparação dos resultados obtidos com abordagens atuais e semelhantes na literatura.

1.4 Estrutura da Dissertação

Para além da introdução, esta Dissertação contém mais seis capítulos. No Capítulo 2, é descrito o estado da arte, onde são apresentados trabalhos relacionados, conjuntos de dados disponíveis, técnicas de extração de características e abordagens MER existentes. No Capítulo 3 é feita uma apresentação detalhada do problema que esta Dissertação pretende resolver, sendo de seguida descrita a solução, assim como os aspetos metodológicos relevantes. No Capítulo 4 é descrita a implementação da abordagem na íntegra, aprofundando todas as arquiteturas de *Machine Learning*, *Deep Learning* e *ensemble learning* empregues. Finalmente, no Capítulo 5, são apresentadas as conclusões da Dissertação e são discutidas potenciais direções de investigação futuras. Por último, são apresentadas às referências e os anexos.

2. Revisão Bibliográfica

2.1 Contexto Histórico

A exploração da ligação entre a música e as emoções tem sido objeto de extensa investigação ao longo dos anos. Uma das pesquisas pioneiras sobre esta relação foi realizada por Hevner em 1936 (Hevner, 1936), na qual identificou 66 adjetivos, posteriormente categorizados em oito grupos distintos, que retratam a essência emocional transmitida na música (ver Figura 1).

Partindo da premissa de Hevner, Russell propôs uma nova perspetiva em 1980, sugerindo que o estado de espírito pode ser quantificado e medido através de um conjunto contínuo de indicadores (Russell, 1980). Russell introduziu o *circumplex model of affect* (ver Figura 2), que define um espaço bidimensional caracterizado pela valência (*valence*) e pelo estímulo (*arousal*). A valência representa a intensidade da emoção, podendo ser positiva ou negativa e variando de desagradável a agradável, e o estímulo denota a intensidade da música, variando de passiva a ativa. Este modelo foi amplamente adotado desde a sua apresentação e continua a ser uma das principais referências no domínio de *MER*.

No início deste século, houve um aumento significativo na investigação dedicada à automação da extração de emoções a partir de informação musical. Em 2007, o *Music Information Retrieval Evaluation eXchange* (MIREX), uma competição internacional sobre recuperação e avaliação de áudio, introduziu o desafio do *Music Emotion Recognition* e incluiu nas suas competições provas de classificação do estado de espírito musical (*Audio Mood Classification - AMC*) (Hu *et al.*, 2008). Posteriormente, Kim e colaboradores em 2010 (Kim *et al.* 2010), efetuaram uma revisão extensa dos trabalhos em *MER*, apresentando uma análise detalhada do tópico.Atualmente, o aparecimento e evolução de aprendizagem profunda (*Deep Learning – DL*) trouxe ao *MER* novos desafios e oportunidades, que constituem uma fase importante no seu desenvolvimento.

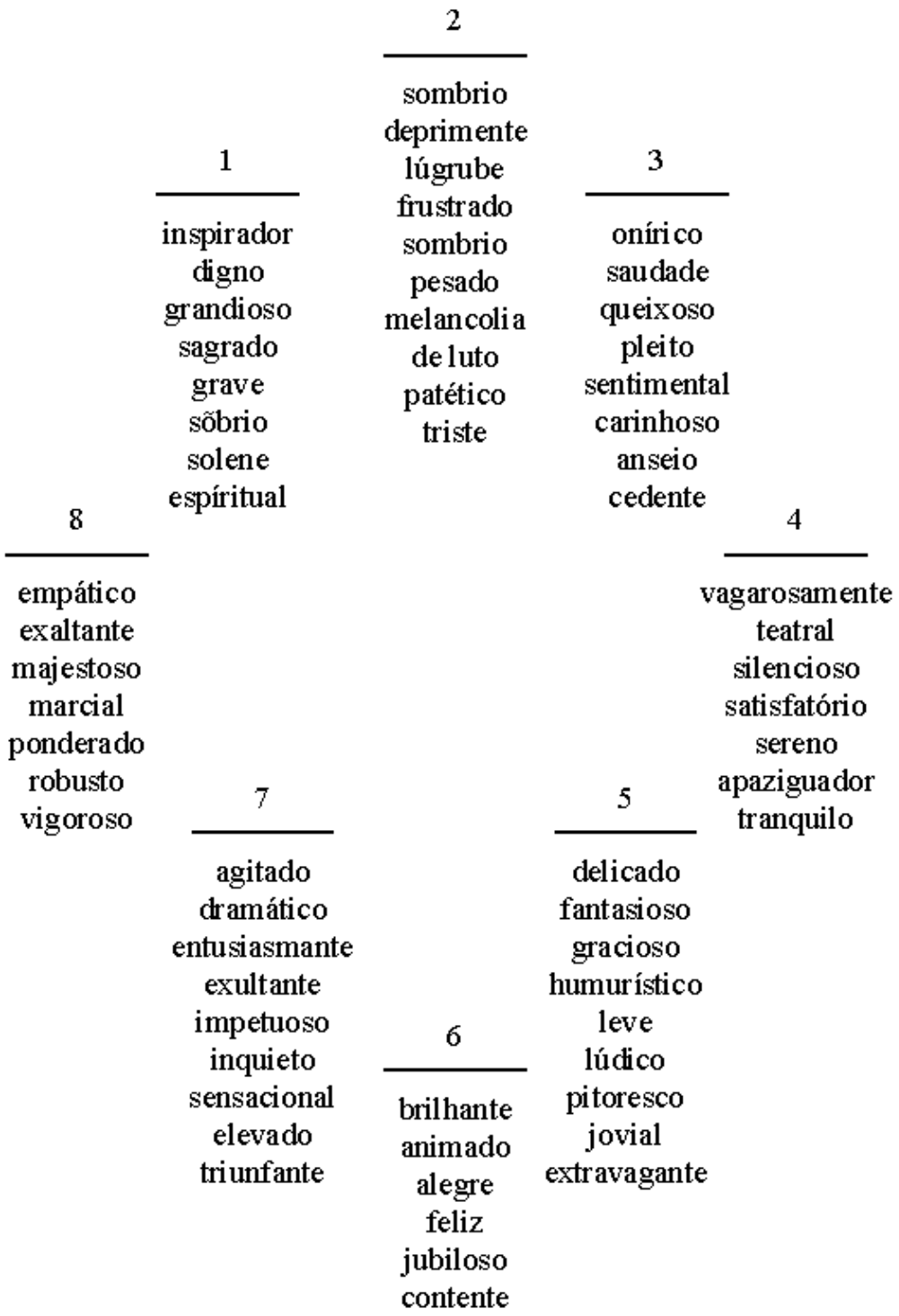


Figura 1: *Cluster* de adjetivos de Hevner: Sessenta e seis palavras organizadas em oito grupos, que descrevem uma variedade de emoções (adaptado de Parisi *et al.* (2019)).

Figura 2: Espaço de Valência-Estímulo (*Valence-Arousal*) de Russell: Um espaço bidimensional que representa os estados de espírito como valores numéricos contínuos. A combinação de valores de valência e estímulo representam diferentes emoções consoante as suas coordenadas no espaço 2D (adaptado de Parisi *et al.* (2019)).

2.2 *Datasets*

A eficácia de um sistema *MER* é diretamente proporcional à qualidade dos dados que este processa. Uma compreensão rigorosa e exata da emoção musical depende da existência de conjuntos de dados (*datasets*) abrangentes e bem organizados. No entanto, devido a restrições de direitos de autor, alguns investigadores de *MER* recorrem à criação dos seus próprios *datasets* que, muitas vezes, não são disponibilizados ao público. Alguns exemplos destes *datasets* podem ser encontrados nos trabalhos de Hu *et al.* (2009), Laurier *et al.* (2008) e Malheiro *et al.* (2018).

Por outro lado, os *datasets* que são disponibilizados ao público são, normalmente, de pequena dimensão e contêm características áudio das músicas, assim como anotações que descrevem as emoções sentidas pelos utilizadores quando ouvem as mesmas. Existem alguns *datasets* de referência, como o *MIREX Mood Dataset*, que contém cerca de 2.000 canções (Downie *et al.* 2008), o *DEAM Dataset* composto por 1.800 canções (Alajanki *et al.* 2016) e o *AllMusic Dataset* com 900 músicas (Korzeniowski *et al.*, 2020). Especificamente, as anotações do *MIREX Mood Dataset* baseiam-se nos *clusters* de adjetivos de Hevner, enquanto as anotações do *DEAM* e do *AllMusic datasets* baseiam-se no modelo de Russell, sendo a principal vantagem destes *datasets* o facto das músicas já estarem emocionalmente anotadas. No entanto, continua em aberto o desafio da disponibilização de *datasets* de grande dimensão com características áudio adequadas e com anotações emocionais que apresentem baixos níveis de subjetividade.

É neste contexto que foi apresentado um dos *datasets* mais interessantes dos últimos anos, com uma quantidade substancial de dados para livre utilização, com informação sobre música e letras em simultâneo: o *WASABI Song Corpus Dataset* (Fell *et al.*, 2022). O projeto *WASABI* (*Web Audio Semantic Aggregated in the Browser for Indexation*) teve início em 2017 e foi criado como resposta às dificuldades existentes no enquadramento de *datasets* previamente mencionadas (Meseguer-Brocal *et al.*., 2017). Este *dataset* contém cerca de dois milhões de canções com letras, anotadas a diferentes níveis e combina meta dados recolhidos a partir de bases de dados musicais na Web, incluindo informações sobre artistas, discografia, produtores, ano de produção, meta dados resultantes da análise das letras das canções: conteúdo das letras, emoções transmitidas pelas mesmas, estrutura das letras e ainda meta dados resultantes da análise do áudio: ritmo, volume, acordes, estrutura, separação de sons, entre outros.

2.3 Extração de Características e Reconhecimento Emocional

Atualmente, existem dois métodos principais para extração de características da música de modo a efetuar reconhecimento emocional: modelos tradicionais de *Machine Leaning* (*ML*), que realizam a extração e o reconhecimento emocional em separado, sendo a extração das características realizada de forma manual; modelos de *Deep Learning* (*DL*), que conseguem realizar ambas as tarefas em simultâneo e de forma automática.

2.3.1 Características Extraídas Manualmente e Modelos Tradicionais de *Machine Learning*

- **Características áudio:** constituem os elementos mais investigados no domínio de *MER*. Estas características são extraídas maioritariamente de ficheiros áudio, através de *toolkits* existentes na literatura (Tzanetakis & Cook, 2000) (Mathieu *et al.*, 2010) (Lartillot *et al.* 2007) (McKay *et al.* 2005). As características áudio relacionadas com as emoções podem ser categorizadas como rítmicas: duração, tom, energia, entre outros. (Han *et al.* 2014), timbre: *Mel-Frequency Cepstral Coefficients* (MFCC), *Zero-Crossing Rate*, entre outros. (Liu *et al.*, 2017) e espectrais: *spectral flatness measure*, *spectral centroid*, entre outros. (Liu *et al.*, 2017).

- **Características simbólicas:** características extraídas de partituras musicais simbólicas, um domínio menos explorado em comparação às características áudio no domínio de *MER*. As partituras simbólicas são normalmente representadas através de *Musical Instrument Digital Interface – MIDI*, uma norma técnica que define um protocolo de comunicação que inclui sequências de tons, tempo, volume e muito mais. As características simbólicas predominantes centram-se à volta de tom, intervalos, intensidade e duração (Chen *et al.* 2016).

- **Características líricas:** desempenham um papel importante na comunicação semântica (Juslin & Laukka, 2004), o que as torna um ponto fundamental na investigação. Na extração manual destas características, os investigadores recorrem principalmente a técnicas de Processamento de Linguagens Naturais (*NLP*) como *Bag-of-Words* (*BOW*) (Yang *et al.* 2004) (He *et al.*, 2008) (Hu *et al.* 2009), *n-grams* (He *et al.*, 2008) e análise *part-of-speech* (*POS*) (Hu *et al.* 2009). Malheiro *et al.* (Malheiro *et al.*, 2018) argumentam que as características convencionais de *BOW* e *POS* podem ser insuficientes

no reconhecimento emocional, propondo a extração de características inovadoras como gíria, semântica e análise estrutural.

- **Características biológicas:** implicam a recolha de dados fisiológicos dos ouvintes. O eletroencefalograma (*EEG*) destaca-se como um método particularmente eficaz para captar informação emocional devido à sua elevada resolução temporal e à sua relação custo-eficácia (Thammasan *et al.*, 2016). Sendo a caraterística biológica mais estudada, o *EEG* tornou-se fundamental na investigação das emoções. Os avanços na tecnologia médica e a oferta de dispositivos portáteis facilitaram a recolha de dados fisiológicos, incluindo a frequência cardíaca e a temperatura da pele, tornando estas variáveis mais visíveis na investigação (Hu *et al.*, 2018). Alguns investigadores também exploraram métodos de extração de dados por indução, através de ressonância magnética funcional (*fMRI*) (Nawa *et al.*, 2018). Apesar de se encontrarem numa fase inicial no seu desenvolvimento em relação às características de áudio e líricas, as características biológicas demonstram um enorme potencial.

2.3.2 Reconhecimento de Emoções Musicais baseado em *Deep Learning*

Desde a sua proposta formal em 2006 (Hinton *et al.*, 2006), *Deep Learning* tem apresentado um desenvolvimento exponencial. Modelos como redes neurais convolucionais (*Convolutional Neural Networks - CNN*) ou redes neurais recorrentes (*Recurrent Neural Networks - RNN*) funcionam como sistemas versáteis de processamento ponta-a-ponta, que permitem que o processo de aprendizagem seja delegado sem quaisquer interrupções pela estrutura de *DL*, que efetua o mapeamento dos resultados em função dos dados originais. Os modelos de reconhecimento de emoções musicais baseados em *Deep Learning* oferecem duas vantagens distintas em relação aos modelos tradicionais de *Machine Learning*: o desempenho destes aumenta em função da quantidade de dados utilizados para treino e também possuem a capacidade de extrair automaticamente características relevantes dos dados fornecidos.

Nas secções seguintes, são aprofundadas as análises relativas a *MER* categórico e *MER* dimensional ao nível da canção. É de notar que os *datasets* disponibilizados ao público, rotulados dinamicamente para o reconhecimento de emoções, inserem-se maioritariamente na categoria dimensional (Thammasan *et al.*, 2016) (Aljanaki *et al.*, 2017) (Speck *et al.* 2011).

2.3.2.1 Reconhecimento de Emoções Musicais Categóricas ao nível da Canção

CNN é um dos algoritmos de aprendizagem de *DL* que permite simular a perceção visual de um ser vivo e aprender representações de características a partir de dados fornecidos. Este algoritmo foi objeto de múltiplos estudos ao longo dos anos, pelo que apresenta variadas estratégias e mutações na literatura.

Liu (2017) interpreta sinais de áudio através de um espetrograma obtido pela transformada de Fourier de curto-termo (*short-time Fourier transform*). O espetrograma de cada música é processado através de uma sequência de camadas convolucionais, culminando em previsões feitas através de uma função *SOFTMAX*, que converte um vetor de *k* números reais numa distribuição de probabilidade de *k* resultados possíveis. A inovação desta investigação é a aplicação de *CNN* para aliviar a carga manual da extração de características, para além de utilizar um método de convolução no tempo que permite uniformizar o comprimento de cada espetrograma. No entanto, este método revela difícil de inferir qual a caraterística que contém as informações pertinentes à emoção musical.

Keelawat *et al.* (2019) incorporaram *EEG* e outras características biológicas em *CNN* para reconhecimento de emoções, produzindo resultados superiores às metodologias tradicionais. Apesar dos resultados positivos obtidos, os autores reconhecem a necessidade de um conjunto de dados mais diversificado para aumentar ainda mais a robustez e a generalização do seu modelo.

Yang *et al.* (2020) exploram vários métodos de extração de características para transformar os dados originais em espectrogramas, alimentando posteriormente a rede neural com estes, de modo a efetuar reconhecimento emocional. Os resultados indicam que os espectrogramas derivados da *Constant-Q Transform* (*CQT*) apresentam o melhor desempenho entre os métodos testados.

2.3.2.2 Reconhecimento de Emoções Musicais Dimensionais ao nível da Canção

As estruturas de *DL* baseadas em *CNN* e *RNN* destacam-se como os modelos mais utilizados. A Rede Neural Recorrente (*Recurrent Neural Network - RNN*), outro algoritmo representativo de *DL*, é excelente no tratamento de dados sequenciais, o que a torna uma escolha comum em aplicações de *NLP*. Variantes como RNN Bidirecional (*Bi-directional RNN - Bi-RNN*) e Memória de Curto-Prazo Longa (*Long Short-Term Memory - LSTM*) são duas adaptações muito utilizadas desta arquitetura.

Ma *et al.* (2017) introduzem o Modelo Baseado no Contexto em Várias Escalas (*Multi-Scale Context-Based Attention Model - MCA*), que inova na integração dinâmica de várias escalas de tempo para compreender a informação temporal e hierárquica da música.

Liu *et al.* (2019) empregam Memória de Curto-Prazo Longa Bidirecional (*Bi-directional Long Short-Term Memory – Bi-LSTM*) para extrair características do áudio. Esta abordagem consiste não só em utilizar *LSTM*, mas também em extrair características constantes, integrando-as antes de introduzir as características combinadas na camada para o reconhecimento emocional. Estas características constantes, obtidas através da informação sobre o ritmo e a energia, contribuem para a originalidade da metodologia.

Dong *et al.* (2019) empregam uma Rede Dispersa Recorrente Convolucional Bidirecional (*Bi-directional Convolutional Recurrent Sparse Network - BCRSN*), uma fusão que combina a aprendizagem adaptativa de *CNN* com a aptidão de *RNN* no processamento de dados sequenciais. O carácter distinto deste trabalho está na integração de *CNN* e *LSTM* para reforçar a capacidade de aprendizagem de características a partir do espetrograma.

Li *et al.* (2016) afirmam que a emoção num ponto específico de uma música não está apenas relacionada com o conteúdo anterior a esse ponto, mas também se propaga para além deste. Em resposta a esta proposta, introduziram a Memória de Memória de Curto-Prazo Longa Bidirecional Profunda (*Deep Bidirectional Long Short-Term Memory - DBLSTM*) para captar informações em ambas as direções temporais. O modelo é composto por três elementos: DBLSTM, pós-processamento e fusão. Através de experiências realizadas no *dataset MediaEval*, descobriram que a utilização de *DBLSTM* por si só ultrapassa o desempenho da Regressão Logística Multinomial (*Multinomial Logistic Regression - MLR*) e da Regressão Vetorial de Suporte (*Support Vector Regression - SVR*), indicando a capacidade de *DBLSTM* para captar informações contextuais. Além disso, a inclusão do pós-processamento e da fusão, especificamente o pós-processamento após a fusão, aumenta a eficácia global, realçando a importância da combinação destes elementos na sua abordagem.

Nas investigações de Devlin *et al.* (2018, 2019), o modelo baseado em transformadores BERT obteve resultados inovadores em múltiplos *benchmarks*, tornando-o uma referência obrigatória em estudos recentes de NLP (Müller *et al.*, 2023) (Perez *et al.* 2022).

No trabalho de Chaki *et al.* (2020), os investigadores introduzem *attentive LSTM*, uma variação de *LSTM* reforçado com um mecanismo de atenção modificado. A sua convicção centra-se na ideia de que a expressão emocional da música num determinado momento depende apenas do contexto musical precedente. Consequentemente, ao calcularem o vetor de contexto para um dado momento, têm exclusivamente em conta os estados que o precedem. Os resultados obtidos indicam uma melhoria notável em comparação à literatura existente (Weninger *et al.*, 2014).

2.4 Conclusão

Estabelecer comparações diretas entre os resultados das propostas apresentadas constitui um enorme desafio, devido ao uso de ferramentas distintas para extração de características, *datasets* anotados com diferentes emoções e métodos de classificação divergentes. Esta dificuldade de comparação é semelhante às conclusões em (Panda *et al.* 2019), que também encontraram desafios na sua análise dos diferentes sistemas de MER. As complexidades na uniformização de metodologias e conjuntos de dados tornam difícil estabelecer paralelismos diretos entre as diferentes investigações.

De um modo geral, o domínio de MER está a transitar de um processamento estático para um processamento dinâmico, de modalidade única para multimodal e de modelos tradicionais de *ML* para modelos de *DL*. Além disso, há espaço para a investigação de fontes de informação mais complexas e detalhadas, como a voz do cantor, *tags* sociais e dados sobre capas de álbuns, expandindo assim as dimensões da investigação.

No futuro, as soluções de Reconhecimento de Emoções Musicais devem abordar os problemas de maior relevo, como a necessidade urgente de *datasets* diversificados em grande escala, assim como modelos de reconhecimento emocional mais fiáveis. Além disso, é necessário integrar conceitos musicais e características cuidadosamente selecionadas nestas soluções, sendo que estas considerações visam aumentar a eficácia e a fiabilidade dos sistemas MER na captação das complexidades da emoção musical.

3. Abordagem Multimodal para Reconhecimento Emocional na Música

3.1 Introdução

Este Capítulo tem como objetivo apresentar a abordagem proposta para o reconhecimento emocional na música e começa por detalhar o problema de investigação a resolver, destacando as limitações das abordagens atuais. De seguida, é apresentada a solução proposta, descrevendo a arquitetura global do sistema e os seus principais componentes. O Capítulo também discute os aspetos metodológicos relevantes, incluindo técnicas de pré-processamento de dados, extração de características e os algoritmos de aprendizagem automática utilizados.

3.2 Definição do Problema

O reconhecimento emocional na música trata-se de um problema complexo, influenciado por múltiplos fatores: a subjetividade das emoções em estudo, a diversidade de características emocionalmente relevantes e ainda a escassez de dados de qualidade.

3.2.1 Subjetividade Emocional

As reações emocionais causadas pela música são extremamente individuais, sendo influenciadas por experiências pessoais, contexto cultural, gosto musical e até estados emocionais temporários. Não obstante, existe uma influência direta dos fatores contextuais da própria música, como as letras ou o ambiente social em que esta é ouvida. Esta ambiguidade complica o processo

de anotação de dados, sendo que diferentes pessoas podem atribuir rótulos contraditórios, tornando difícil a avaliação do desempenho dos sistemas MER.

3.2.2 Multiplicidade de características relevantes

A um nível basilar, as emoções que a música carrega são moldadas por características áudio simples como frequência, timbre, ritmo e harmonia. De uma perspetiva hierárquica, é possível argumentar que as letras desempenham um papel significativo na formação do significado emocional após estas características áudio, uma vez que expressam ou sugerem emoções específicas, enquanto expõe simultaneamente a narrativa e os temas gerais da música. Esta multiplicidade de características representa um desafio para os sistemas MER, uma vez que a captação e integração destas características exigem técnicas sofisticadas de extração e aprendizagem de representações que consigam lidar com a natureza multimodal da música. As abordagens mais tradicionais, como modelos de Regressão Linear ou *Random Forest* podem apresentar dificuldades em captar as interações complexas entre estas modalidades. Já as arquiteturas mais complexas de *Deep Learning*, como redes neurais convolucionais (CNN) ou redes neuronais recorrentes (RNN), apresentam um potencial promissor e uma capacidade superior na aprendizagem com dados áudio e de texto.

3.2.3 Escassez de dados rotulados

A música é um material protegido por direitos de autor, sendo a distribuição da mesma rigidamente regulada, o que dificulta o acesso e a partilha de grandes coleções de ficheiros para efeitos de treino e avaliação de modelos MER. Embora tenham sido criados alguns conjuntos de dados através de parcerias entre investigadores e intervenientes da indústria musical, estes são geralmente limitados na sua dimensão e nos géneros musicais que abrangem. Isto, aliado à natureza subjetiva da música, dá origem a ruído nos dados recolhidos e a erros na classificação.

3.2.4 Limitações das abordagens

As abordagens tradicionais para reconhecimento emocional na música baseiam-se em características manualmente extraídas e em algoritmos de *Machine Learning,* que apresentam várias limitações na sua capacidade de compreensão emocional. Estes algoritmos funcionam normalmente através de vetores de características com um comprimento fixo que não apresentam capacidade de compreensão da dinâmica temporal e das dependências a longo prazo da música.

Devido a estas limitações, tem havido um crescente interesse nas abordagens de *Deep Learning* que conseguem identificar e extrair automaticamente características relevantes, lidar com múltiplas modalidades e ainda treinar com dados não rotulados. No entanto, estas abordagens também apresentam os seus próprios desafios, como a necessidade de grandes recursos computacionais, o risco de sobreajuste e a dificuldade de interpretação das representações obtidas.

3.3 Solução Proposta

De modo a enfrentar os desafios discutidos, é proposta uma abordagem multimodal para o reconhecimento emocional na música, que tem como objetivo reunir as vantagens das técnicas tradicionais de *Machine Learning* e das técnicas avançadas de *Deep Learning,* através do treino de múltiplos modelos nas características que mais se adequam à arquitetura em questão. Os melhores resultados de ambas as modalidades são posteriormente agregados de modo a criar um modelo de fusão.

Para as informações de áudio, são empregues algoritmos de *Machine Learning*, como Regressão Linear, *Random Forest*, *Gradient Boosting* e Máquina de Vetores de Suporte (SVM). Estes algoritmos apresentam uma elevada capacidade no tratamento de dados estruturados, como características numéricas e categóricas, sendo por isso escolhidos para treinar num conjunto de características áudio como o tempo (BPM), o ganho, a duração e ainda metadados dos acordes (estes englobam informação sobre os acordes utilizados, o número de acordes e as respetivas durações).

Para o estudo da informação lírica são empregues modelos de *Deep Learning* como Redes Neurais Convolucionais (CNN), redes de Memória de Curto Prazo Longo (*Long Short Term Memory*: LSTM), LSTM Bidirecional (BiLSTM) e Redes de Atenção Hierárquica (*Hierarchical Attention Network*: HAN). Estes modelos são particularmente adequados para o processamento de dados sequenciais, como texto, e permitem captar a informação semântica e contextual presente das letras.

De modo a melhorar a aprendizagem das características líricas, são incorporados modelos BERT pré-treinados em cada uma das arquiteturas de *Deep Learning*. Como já foi referido em capítulos anteriores, o BERT trata-se de um modelo de linguagem que revolucionou as tarefas de processamento de linguagem natural através da aprendizagem de representações bidireccionais de texto. Estes modelos foram treinados em grandes quantidades de dados de texto, o que lhes permite compreender a semântica da linguagem e captar o conteúdo emocional expresso nas letras.

A representação e categorização das emoções nesta investigação baseia-se no espaço de emoção valência-estímulo, inspirado no modelo circumplexo de afeto de Russell. A valência representa a intensidade da emoção, sendo positiva ou negativa e variando de desagradável a agradável, enquanto o estímulo representa o nível de energia ou intensidade da emoção e varia de passiva a ativa. As anotações de valência e estímulo utilizadas para o treino dos diferentes modelos são fornecidas pelo conjunto de dados WASABI.

3.4 Conclusão

Neste capítulo, foram abordados os diversos desafios que existem no ramo de MER, sendo posteriormente proposta uma abordagem multimodal para enfrentá-los. As principais dificuldades incluem a subjetividade das respostas emocionais à música, a diversidade de características emocionalmente relevantes e a escassez de dados de qualidade.

A solução proposta combina algoritmos tradicionais de *Machine Learning* e técnicas avançadas de *Deep Learning* para aprendizagem de diferentes modalidades. Para informações de áudio são utilizados algoritmos como Regressão Linear, *Random Forest*, *Gradient Boosting* e *Support Vetor Machines* (SVM), treinados em características comol BPM, ganho, duração e informações relacionadas com acordes, tirando partido da capacidade que apresentam de lidar com dados estruturados e numéricos.

Para a informação lírica, são utilizados modelos de *Deep Learning* como Redes Neurais Convolucionais (CNN), Redes de Memória de Curto Longo Prazo (LSTM), LSTM bidirecional (BiLSTM) e Redes de Atenção Hierárquica (HAN). Estes modelos são eficazes no processamento de dados de texto sequenciais e na captação da informação semântica e contextual. De modo melhorar esta aprendizagem, são incorporados modelos BERT pré-treinados em todas as arquiteturas referidas, uma vez que estes apresentam um desempenho elevado em tarefas de processamento de linguagem natural.

Por último, de modo a aproveitar os pontos fortes de diferentes modelos e melhorar o desempenho global de previsão, é implementado um modelo de fusão através da junção dos resultados obtidos pelos modelos com melhor desempenho de cada uma das modalidades áudio e lírica.

A natureza multimodal da abordagem apresentada permite a integração de informações complementares de características de áudio e letras, juntamente com o espaço de valência-estímulo e modelos BERT pré-treinados. Nos capítulos seguintes são apresentados detalhes sobre a implementação, o conjunto de dados utilizado, as métricas de avaliação e os resultados experimentais obtidos.

4. Implementação

4.1 Introdução

Este Capítulo tem como objetivo descrever a implementação da abordagem proposta pela investigação, começando por apresentar o conjunto de dados utilizado, as técnicas de pré-processamento e a preparação dos dados para as fases subsequentes. De seguida, é descrita a arquitetura de todos os modelos implementados, abrangendo os modelos de *Machine Learning*, *Deep Learning* e fusão. Para cada modelo, são fornecidos detalhes sobre a sua estrutura, hiperparâmetros e processo de treino. O Capítulo também inclui uma discussão sobre as métricas de avaliação utilizadas para calcular o desempenho dos modelos.

4.2 Abordagem ao Conjunto de Dados

O conjunto de dados WASABI trata-se de um conjunto abrangente de canções enriquecido com metadados de várias bases de dados musicais na Web. É construído a partir da *LyricsWikia* e contém 1,73 milhões de canções com letras, anotadas a diferentes níveis, incluindo segmentação estrutural, tópicos, caráter explícito, passagens salientes e emoções transmitidas. Inclui também metadados para 2 milhões de canções, 77 mil artistas e 200 mil álbuns, juntamente com ligações e identificações para outras plataformas, como a Wikipédia, o *YouTube*, o *MusicBrainz*, o *Last.fm* e o *Discogs* (Fell *et al.*, 2022).

De modo a extrair informações relevantes das letras, o conjunto de dados WASABI incorpora vários métodos de processamento de linguagem natural. Estes métodos incluem:

- Segmentação estrutural: treino de uma rede neural convolucional que prevê as fronteiras dos segmentos nas letras das músicas a partir de matrizes de auto-similaridade (SSM) que

codificam a sua estrutura repetitiva. Assim, as canções são associadas a segmentos de texto etiquetados que correspondem a versos, refrões, introduções, etc (Fell et al., 2018).

- Tópicos: construção de um modelo de tópicos sobre as letras, utilizando *Latent Dirichlet Allocation* (LDA). Estes tópicos podem ser visualizados através de "*word clouds*" com as palavras mais significativas de cada tópico (Fell et al., 2019) (Fell, 2020).
- Deteção de conteúdo lírico explícito: comparação de métodos automatizados, desde a pesquisa baseada em dicionários até a *deep neural networks* avançadas para deteção automática dos conteúdos explícitos nas letras de música (Fell, Cabrio, Corazza, et al., 2019).
- Passagens salientes de uma canção: aplicação de método que permite resumir as letras através da relação entre o áudio e a letra (*audio thumbnailing approach*) (Fell, Cabrio, Gandon, et al., 2019).
- Emoções transmitidas: treino de um modelo de regressão de emoções utilizando o BERT para classificação das emoções com base no modelo valência-estímulo (Fell et al., 2019).

Uma vez que o conjunto de dados escolhido para esta investigação foi o WASABI, tal como este, utiliza o modelo valência-estímulo para categorização emocional, que se alinha com o modelo circumplexo de afeto de Russell.

4.2.1 Preparação dos Dados

Devido ao tamanho excessivo do conjunto de dados original (12 GB), foi necessário realizar um pré-processamento ao mesmo de modo a extrair apenas as informações relevantes. Utilizando um script personalizado, foi criado um novo *dataset*, denominado "*song_extracted_all_entries.json*", que engloba todas as características essenciais para treinar os modelos num conjunto de dados pré-processado de 5 GB. Na Figura 3 é apresentada uma das entradas deste novo conjunto de dados.

{"_id": "5714dec325ac0d8aee3804ec", "lyrics": "Everybody in the building has to sing the song All the boys and all the girlfriends sing the sing-a-long Think I'm in trouble, there's always a couple Around me wherever I go They're out there to bug me I don't think it's funny Everybody's laughing at me, yeah I wanna go out but there's no-one about All my friends want a quiet one at home The same age as me and They're husbands-to-be, yeah Everybody in the building Everybody in the building has to sing the song All the boys and all the girlfriends sing the sing-a-long I make a move here and I make a move there, I've got millions of things on the go I write me the best lines They're too corny sometimes Everybody's laughing at me Everybody in the building has to sing the song All the boys and all the girlfriends sing the sing-a-long Everybody in the building has to sing the song All the boys and all the girlfriends sing the sing-a-long I know when I find her There's rings on her finger I've tried every trick in the book I've got some hard pride I've got time on my side Everybody's gone without me, yeah Everybody in the building has to sing the song All the boys and all the girlfriends sing the sing-a-long Everybody in the building has to sing the song All the boys and all the girlfriends sing the sing-a-long", "bpm": "97.3", "gain": "-12.4", "length": "44", "chords_metadata": {"confidence": 0.8390022675736961, "duration": 44.01739229024943, "chordSequence": [{"start": 0.0, "end": 6.050000000000001, "label": "Cmaj"}, {"start": 6.050000000000001, "end": 9.75, "label": "Fmaj"}, {"start": 11.350000000000001, "end": 17.150000000000002, "label": "Fmaj"}, {"start": 17.150000000000002, "end": 17.750000000000004, "label": "Cmaj"}, {"start": 17.750000000000004, "end": 18.950000000000003, "label": "Bbmaj"}, {"start": 18.950000000000003, "end": 26.450000000000003, "label": "Fmaj"}, {"start": 26.450000000000003, "end": 27.950000000000003, "label": "Cmaj"}, {"start": 27.950000000000003, "end": 33.75, "label": "Fmaj"}, {"start": 33.75, "end": 34.349999999999994, "label": "Cmaj"}, {"start": 34.349999999999994, "end": 35.65, "label": "Bbmaj"}, {"start": 35.65, "end": 41.45, "label": "Fmaj"}, {"start": 41.45, "end": 44.0, "label": "Fmaj7"}], "_id": "deezer:12675227"}, "arousal": "", "arousal_predicted": "0.25471792", "valence": "", "valence_predicted": "0.2352858"}

Figura 3: Amostra de uma entrada na base dados criada "*song_extracted_all_entries.json*".

4.3 Arquitetura dos Modelos

A seleção das características mais apropriadas para treinar cada um dos modelos de *Machine Learning* e *Deep Learning* baseou-se nos pontos fortes de cada uma destas diferentes abordagens. Os modelos de *Deep Learning* apresentam uma capacidade elevada de aprendizagem e extração de padrões complexos para dados não estruturados, sendo particularmente adequados na análise e avaliação do conteúdo emocional das letras das canções. Os modelos de *Machine Learning* são altamente eficazes no tratamento de dados estruturados, como características numéricas e categóricas, sendo por isso utilizados para treinar em características como BPM, ganho, duração e ainda os metadados dos acordes (que abrange os acordes utilizados em cada canção, o número de acordes e a duração de cada um).

Como é possível observar na Figura 3, cada uma das entradas na base de dados apresenta valores de valência e estímulo únicos, obtidos através dos diversos métodos de processamento referidos na secção 4.1. Estes valores são utilizados em conjunto com as características pertinentes a cada um dos modelos de modo a obter novos valores valência e estímulo.

O desempenho de cada modelo foi estudado utilizando três métricas de avaliação: *Mean Absolute Error* (Erro Absoluto Médio) (MAE), *Mean Squared Error* (Erro Quadrático Médio) (MSE) e Coeficiente de determinação (R^2).

MAE mede a diferença média absoluta entre os valores previstos e reais, fornecendo uma compreensão do erro de previsão do modelo. É calculado da seguinte forma:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{4.1}$$

onde:

- n é o número de amostras.
- y_i é o valor real.
- x_i é o valor previsto

O MSE, calcula a diferença média ao quadrado entre os valores previstos e reais, dando uma maior importância a erros de maior dimensão. É representado da seguinte forma:

$$MSE = \frac{\sum_{i=1}^{n} |y_i - x_i|^2}{n} \tag{4.2}$$

O coeficiente de determinação é uma medida estatística que determina a proporção da variação na variável dependente que pode ser explicada pela variável independente. Ou seja, o coeficiente de determinação mostra se os dados se ajustam ou não ao modelo em questão. Este valor varia entre 0 e 1, com valores mais altos indicando um melhor ajuste. É calculado da seguinte forma:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{4.3}$$

onde:

- SS_{res} é a soma dos quadrados residuais. Mede o desvio total dos valores previstos em relação aos valores observados.
- SS_{tot} é a soma total dos quadrados. Representa a variação total da variável dependente (no caso da presente investigação, os valores de valência e estímulo).

4.3.1 *Machine Learning*

4.3.1.1 Regressão Linear

O modelo de Regressão Linear serve como modelo base para esta investigação, fornecendo uma referência para o desenvolvimento de modelos subsequentes. A regressão linear é um algoritmo simples, mas eficaz, que assume uma relação linear entre as características de entrada e as variáveis-alvo.

O modelo de regressão linear criado utiliza então seis características de entrada: 'bpm', 'gain' (ganho), 'length' (duração da canção), 'chord_labels' (os acordes existentes na canção), 'num_chords' (número de acordes), 'avg_chord_duration' (duração média de cada acorde). Estas características captam simultaneamente aspetos das características musicais e da estrutura das canções.

Implementação

O modelo consiste em duas ocorrências separadas de Regressão Linear, uma para prever a valência e outra para prever o estímulo. A arquitetura do modelo de Regressão Linear pode ser representada por:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \quad (4.4)$$

onde:

- y representa a variável alvo (valência ou estímulo).
- β_0 é o termo de interceção.
- β_1, β_2, ..., β_n são os coeficientes correspondentes a cada elemento de entrada.
- x_1, x_2, ..., x_n são os elementos de entrada ("bpm", "gain", "length", "num_chords", "avg_chord_duration").

O modelo aprende os valores ideais para os coeficientes (β_0, β_1, ..., β_n) durante o processo de treino, reduzindo a soma dos erros quadráticos entre os valores previstos e os valores reais.

Os hiperparâmetros do modelo de Regressão Linear são os seguintes:

- fit_intercept: Booleano que indica se deve ser calculada a interceção para o modelo. É definido como *True* por defeito, o que significa que o modelo irá estimar o termo de interceção.
- normalize: Booleano que indica se as características de entrada devem ser normalizadas. É definido como *False* por padrão, indicando que os recursos de entrada não são normalizados antes de ajustar o modelo.
- copy_X: Booleano que indica se as características de entrada devem ser copiadas. É definido como *True* por padrão, garantindo que os recursos de entrada originais não sejam modificados durante o processo de ajuste do modelo.
- n_jobs: Número de trabalhos a utilizar para o cálculo. Por predefinição, está definido como *None*, o que significa que o cálculo é efetuado com um único processador.

O modelo de Regressão Linear serve como uma abordagem simples para prever os valores de valência e estímulo. A sua simplicidade e eficiência computacional tornam-no um ponto de partida forte para explorar a relação entre as características de entrada e os atributos emocionais. No entanto, esta arquitetura pressupõe uma relação linear entre as características e as variáveis alvo, o que pode nem sempre captar as relações complexas e não lineares presentes nas tarefas de reconhecimento emocional.

4.3.1.2 *Random Forest*

O modelo *Random Forest* é um algoritmo de aprendizagem que combina várias árvores de decisão, em que cada árvore divide recursivamente o espaço de entrada com base nos valores das características de modo a efetuar previsões. Baseando-se no modelo de regressão linear, introduz uma arquitetura mais complexa e flexível, com capacidade de captar relações não lineares. A previsão final é obtida pela agregação das previsões de todas as árvores individuais, sendo esta normalmente efetuada através da média das previsões para tarefas de regressão.

As principais modificações efetuadas na criação do modelo *Random Forest* foram as seguintes:

- Aprendizagem *ensemble*: O modelo *Random Forest* utiliza um conjunto de árvores de decisão, em que cada árvore é treinada num subconjunto aleatório dos dados de treino e num subconjunto aleatório das características de entrada. Esta abordagem ajuda a reduzir *overfitting* (ajuste) e a melhorar o desempenho da generalização.
- Alteração dos hiperparâmetros:
    1. n_estimators: O número de árvores é definido para 100, o que determina o tamanho do conjunto. Aumentar este número pode melhorar o desempenho, mas também aumenta a complexidade computacional.
    2. max_depth: A profundidade máxima de cada árvore de decisão é definida como 10, limitando a profundidade até que cada uma destas pode crescer. Isto também ajuda a evitar *overfitting* e controla a complexidade de cada árvore.
    3. random_state: A semente aleatória é definida como 42 de modo a garantir a reprodutibilidade dos resultados.

Embora o modelo *Random Forest* ofereça várias vantagens em relação ao modelo de Regressão Linear, como a deteção de relações não lineares e a redução do *overfitting*, também apresenta alguns inconvenientes. Para além de ser um modelo mais dispendioso do ponto de vista computacional, a sua capacidade de interpretação é ligeiramente reduzida comparativamente à regressão linear, uma vez que as árvores de decisão individuais poderem ter estruturas complexas e tomar decisões com base em diferentes subconjuntos de características. Cada árvore pode captar diferentes padrões e relações nos dados e quando são combinadas para realizar a previsão final, torna-se mais difícil de identificar e compreender como é que cada caraterística individual contribui para a previsão global.

4.3.1.3 *Gradient Boosting*

O modelo *Gradient Boosting* é outro algoritmo que se baseia nos conceitos de árvores de decisão, mas também em *boosting* para criar um modelo flexível para tarefas de regressão. Combina vários *weak learners* (árvores de decisão) de forma sequencial para melhorar progressivamente o desempenho do modelo e assim reduzir os erros de previsão.

A ideia principal dos modelos de *Gradient Boosting* é o treino iterativo de uma série de árvores de decisão, no qual cada árvore subsequente é treinada de modo a corrigir os erros cometidos pelas árvores anteriores. O modelo começa com uma previsão inicial, normalmente a média da variável-alvo, e em cada iteração que segue uma nova árvore de decisão é treinada com base nos valores residuais (as diferenças entre os valores reais e os valores previstos). As previsões da nova árvore são adicionadas às previsões existentes e o processo é repetido durante um número especificado de iterações.

Este modelo introduz vários hiperparâmetros que podem ser ajustados de modo a otimizar o seu desempenho:

- n_estimators: Determina o número de iterações ou etapas de reforço (*boosting*). Para efeitos desta investigação, é definido como 100, o que indica que 100 árvores de decisão são sequencialmente treinadas.
- learning_rate: Controla a contribuição que cada árvore tem para a previsão final. Uma taxa de aprendizagem menor (definida no modelo desenvolvido como 0,1) significa que cada árvore tem um impacto menor na previsão geral, permitindo um processo de aprendizagem mais gradual e controlado.
- max_depth: Limita a profundidade máxima de cada árvore. Ao definir max_depth como 3, o *script* restringe a complexidade das árvores individuais, o que evita o ajuste excessivo e promove um modelo mais generalizado.
- random_state: Como no caso do modelo anterior, garante a reprodutibilidade dos resultados ao definir uma semente aleatória fixa.

O processo de treino envolve o ajuste dos dados de treino através do método fit(). O modelo é treinado separadamente para cada uma das dimensões de valência e estímulo, o que permite captar os padrões e relações de cada uma das dimensões. Depois, o modelo combina as previsões de todas as árvores de decisão treinadas para gerar os valores finais através do método predict(), que permite obter os valores previstos de valência e estímulo para o conjunto de teste.

Uma das vantagens da arquitetura *Gradient Boosting* é a capacidade de que apresenta de captar relações não lineares entre as características de entrada e as variáveis-alvo. Ao corrigir

iterativamente os erros das árvores anteriores, o modelo consegue aprender padrões complexos e fazer previsões precisas. Além disso, é um modelo relativamente robusto em relação a *outliers* e pode lidar com valores em falta nas características de entrada.

No entanto, tal como o modelo *Random Forest*, a interpretabilidade do modelo é ligeiramente reduzida em comparação com modelos mais simples como o de Regressão Linear. A previsão final é o resultado das contribuições combinadas de várias árvores de decisão, o que torna mais difícil interpretar diretamente o impacto das características individuais no resultado previsto.

4.3.1.4 Regressão de Vetores de Suporte: *Support Vector Regression*

Support Vector Regression (SVR) é uma variante do algoritmo de *Support Vector Machine* (SVM) que consegue lidar com relações não lineares entre características e variáveis alvo, sendo o principal objetivo encontrar um hiperplano num espaço de dimensão elevada que minimize o erro de previsão e maximize a margem entre o hiperplano criado e os dados.

Para efeitos desta investigação, são treinados dois modelos SVR distintos para prever as dimensões de valência e estímulo. Estes modelos são implementados usando a biblioteca PyTorch, que fornece uma interface de alto nível para construir e treinar estas arquiteturas.

A arquitetura dos modelos SVR é definida pela classe SVRModel, que é herdada da classe nn.Module do PyTorch. O modelo SVRM é composto por uma única camada linear (nn.Linear) que relaciona as características de entrada com as previsões de saída. A dimensão de entrada (input_dim) é determinada pelo número de características nos dados de treino (bpm, ganho, comprimento, num_chords e avg_chord_duration).

Os modelos SVR apresentam vários hiperparâmetros que podem ser ajustados para otimizar o seu desempenho:

- Kernel: Transforma os dados fornecidos num espaço de características de dimensão superior, o que permite ao modelo captar relações não lineares entre as características e as variáveis-alvo. A escolha da função de kernel determina o tipo de transformações aplicadas aos dados sendo que, no modelo criado, é utilizado o kernel da função de base radial (RBF), uma escolha popular para modelos SVR. Outros tipos de kernel incluem kernels lineares, polinomiais e sigmóides.
- Parâmetro de regularização (C): Controla o compromisso entre obter um erro de treino baixo e uma complexidade de modelo baixa. Um valor menor de C permite uma maior regularização e pode ajudar a evitar o ajuste excessivo.

- Epsilon (ε): Define a margem de tolerância onde não é aplicada qualquer penalização aos erros. Determina a largura da zona insensível relativamente ao valor previsto.

É utilizado o optimizador Adam para atualizar os parâmetros do modelo com base nos gradientes calculados, sendo este processo de treino executado para um número de 100 épocas (*epochs*), com um tamanho de grupo (*batch*) de 1024.

Como referido anteriormente, o SVR tem a vantagem de captar relações não lineares complexas entre as características e as variáveis-alvo, o que possibilita a modelação de padrões complexos nos dados que não conseguem ser facilmente interpretados por modelos lineares. Também apresenta uma elevada robustez face a valores atípicos, sendo o parâmetro epsilon aplicado para definir uma margem de tolerância, na qual os dados dentro dessa margem não influenciam a qualidade do modelo. Por último, apresenta a capacidade de lidar com um número elevado de características sem ter de calcular explicitamente as suas coordenadas.

No entanto, também apresenta as suas limitações. Tal como os modelos de *Gradient Boosting* e *Random Forest*, a sua interpretabilidade é reduzida quando comparada com modelos mais simples como Regressão Linear. O limite de decisão aprendido por estes modelos baseia-se num subconjunto de pontos de treino (vetores de suporte) e na função kernel, o que torna mais difícil interpretar diretamente o impacto de características individuais no resultado previsto. Por outro lado, o treino destes modelos também pode ser computacionalmente dispendioso, especialmente quando se trata de um conjunto grande de dados, como é o caso da presente investigação. Por último, os modelos SVR apresentam vários hiperparâmetros que precisam de ser ajustados para obter um desempenho otimizado, sendo necessário experimentação extensiva para encontrar a melhor combinação entre destes.

4.3.2 *Deep Learning*

4.3.2.1 CNN + Mecanismo de Atenção

O modelo CNN + Mecanismo de Atenção é uma arquitetura *Deep Learning* que utiliza o poder das Redes Neurais Convolucionais (CNNs) para extração de características, aliada a um mecanismo de atenção, que possibilita um foco superior nas sequências de entrada mais relevantes.

A componente CNN é responsável pela extração de características relevantes das letras através da aplicação de filtros convolucionais que captam padrões e relações locais. Estes filtros aprendem a detetar características e padrões específicos em várias granularidades, como palavras individuais, frases ou contextos maiores. Estas camadas são normalmente seguidas por camadas de *pooling*, que reduzem a amostragem das características e ajudam a reduzir as dimensões da representação, que permite o foco do modelo nas características mais relevantes.

O mecanismo de atenção trata-se de uma técnica que permite ao modelo se concentrar seletivamente em diferentes partes da sequência introduzida. Neste tipo de arquitetura, o mecanismo de atenção é aplicado no *output* das camadas CNN para determinação da importância de cada palavra ou frase na letra da música. São atribuídos pesos específicos a cada um dos elementos da sequência com base na sua relevância, tendo o modelo a capacidade de aprender quais as partes mais relevantes da letra que contribuem para a previsão dos valores de valência e estímulo.

As letras são pré-processadas e "tokenizadas" utilizando o BERT *tokenizer*, sendo estas palavras "tokenizadas" introduzidas depois num modelo BERT que foi pré-treinado num grande conjunto de dados de texto ("bert-base-uncased") de modo a obter palavras contextualizadas.

A arquitetura do modelo CNN + Mecanismo de Atenção pode ser resumida da seguinte forma:

- Camada BERT: As letras "tokenizadas" são processadas pelo modelo BERT de modo a obter a contextualização das palavras. O último estado do modelo BERT, correspondente ao *token* [CLS], é utilizado como representação completa de toda a sequência.
- Camadas Totalmente Conectadas: O output da camada BERT é transferido para duas camadas totalmente conectadas (fc1 e fc2), sendo estas utilizadas para reduzir as dimensões dos *embeddings* BERT e mapear os valores previstos de valência e estímulo. A primeira camada (fc1) apresenta uma dimensão de 256, que permite ao

modelo realizar o processo de aprendizagem através de uma representação comprimida dos *embeddings*. A segunda camada (fc2) apresenta uma dimensão de 2, que corresponde aos valores de valência e estímulo a prever. Esta redução da dimensão ajuda a evitar *overfitting* e permite que o modelo generalize os dados de uma forma mais eficaz.

Os hiperparâmetros correspondentes ao modelo CNN + Mecanismo de Atenção são os seguintes:

- "hidden_dim": A dimensão da camada oculta na camada totalmente ligada (fc1). Está definida como 256 e controla a capacidade do modelo aprender padrões e relações complexas nos dados.
- "output_dim": A dimensão da camada de saída (fc2) definida como 2, que corresponde aos valores previstos de valência e estímulo.
- "learning_rate": A taxa de aprendizagem para o optimizador AdamW, definida como 1^{-5} (0,00001). Uma taxa de aprendizagem mais baixa garante atualizações graduais dos valores do modelo durante o processo de treino.
- "num_epochs": O número de épocas de treino, definido como 10. Aumentar este número pode melhorar o desempenho do modelo, mas também aumenta o tempo de treino.
- "batch_size": O número de amostras processadas em cada iteração de treino, definido como 32. Um valor superior pode acelerar o treino, mas também requer mais memória.
- "accumulation_steps": O número de etapas de acumulação do gradiente, definido como 4. Permite que o modelo acumule gradientes em vários conjuntos antes de executar uma etapa de otimização, o que possibilita uma gestão eficiente das restrições de memória.

Este modelo beneficia de uma aprendizagem por transferência, o que permite tirar partido do modelo BERT pré-treinado e possibilita a aprendizagem de padrões linguísticos, semântica e relações contextuais. Esta abordagem reduz a necessidade de uma quantidade elevada de dados de treino para que o modelo atinja um desempenho superior em comparação a um treino realizado de raiz.

A principal desvantagem deste modelo é a sua complexidade sendo que, a utilização do modelo BERT, uma rede neural de grande dimensão, requer recursos computacionais significativos e um elevado tempo de treino. A arquitetura do modelo, com várias camadas e mecanismos de atenção, aumenta esta carga computacional e torna o processo mais lento e intensivo comparativamente a modelos mais simples, como os abordados na secção anterior.

4.3.2.2 Long Short-Term Memory

O modelo *Long Short-Term Memory* combina o poder de um modelo BERT pré-treinado com uma rede LSTM para modelar dependências sequenciais nas letras. Os *embeddings* do modelo BERT pré-treinado permitem capturar a informação contextual, enquanto a camada LSTM processa sequencialmente estes *embeddings* e possibilita que o modelo capte as suas dependências temporais.

Tal como o modelo anterior, as letras das músicas são utilizadas como *input*, sendo pré-processadas e "tokenizadas" por um BERT *tokenizer* ("bert-base-uncased"). Estas são posteriormente enviadas para uma camada de incorporação (*embedding*), de modo a obter representações vetoriais para cada *token*, sendo que estes vetores captam o significado semântico e a informação contextual da letra.

A arquitetura do modelo é a seguinte:

- Camada de Incorporação (*embedding*): As letras "tokenizadas" atravessam esta camada de modo a obter representações vetoriais para cada um dos *tokens* criados.
- Camada LSTM: A sequência *token embeddings* é introduzida numa camada LSTM que processa sequencialmente a informação e capta as dependências a longo prazo das letras. Esta camada consiste em várias células LSTM, cada uma com uma dimensão de 256.
- Camada Totalmente Conectada: O último estado da camada LSTM, que representa o contexto geral e o conteúdo emocional das letras, é passado por uma camada totalmente conectada (fc) com uma dimensão de saída de 2, correspondente aos valores a prever de valência e estímulo.

Os hiperparâmetros do modelo são idênticos ao modelo CNN + Mecanismo de Atenção, sendo a única diferença a dimensão da camada oculta, que está associada a uma camada LSTM, com o mesmo valor de 256.

Tal como o modelo CNN + Mecanismo de Atenção, o modelo LSTM beneficia da capacidade de compreensão contextual fornecida pelo modelo BERT pré-treinado. Para além disso, também consegue modelar as dependências sequenciais e a progressão emocional das letras através da camada LSTM, que processa sequencialmente os *tokens embeddings*. Esta modelação sequencial permite considerar o contexto emocional das letras e assim aumentar a precisão da previsão emocional.

As desvantagens do modelo LSTM são semelhantes às do modelo CNN + Mecanismo de Atenção. Sendo ambos arquiteturas de *Deep Learning*, não possuem a capacidade de interpretação

dos modelos mais simples, como o de Regressão Linear, sendo que as interações complexas entre os *embeddings* BERT, a camada LSTM e a camada totalmente ligada tornam difícil compreender como é que o modelo chega às previsões que apresenta. Para além disto, sofre também de uma complexidade computacional elevada, que torna o processo mais lento e intensivo em termos de recursos necessários.

4.3.2.3 BiLSTM + Mecanismo de Atenção

O modelo BiLSTM + Mecanismo de Atenção utiliza redes *Bidirectional Long Short-Term Memory* (BiLSTM) de modo a captar dependências sequenciais, juntamente com um mecanismo de atenção que permite o foco nas partes mais relevantes da sequência fornecida.

Tal como os modelos anteriores, o modelo BiLSTM + Mecanismo de Atenção recebe as letras das músicas como *input*, aplica um pré-processamento e "tokenização" nas mesmas através do BERT *tokenizer*, e introduz a informação processada no modelo BERT pré-treinado ("bert-base-uncased") de modo a obter as palavras contextualizadas.

A arquitetura do modelo BiLSTM + Mecanismo de Atenção é a seguinte:

- Camada BERT: como nos modelos anteriores, as letras "tokenizadas" são passadas através do modelo BERT de modo a obter palavras contextualizadas. No entanto, em vez de utilizar o último estado oculto correspondente ao *token* [CLS], este modelo utiliza os estados ocultos correspondentes a cada *token* na sequência de entrada, o que permite captar informação de uma forma mais refinada.
- Camada BiLSTM: As palavras contextualizadas pela camada BERT são introduzidas na camada BiLSTM, que consiste em duas redes LSTM: uma que processa a sequência posterior e outra que processa a sequência anterior. Ao considerar simultaneamente o contexto passado e futuro da sequência, esta camada consegue captar relações a longo prazo e compreender o contexto global da letra.
- Camada de Atenção: Os resultados da camada BiLSTM passam pelo mecanismo de atenção, que atribui pesos distintos a diferentes partes da sequência e permite um foco superior nas palavras e frases mais relevantes.
- Camada Totalmente Conectada: As características selecionadas pela camada de atenção são transferidas para uma camada totalmente conectada que relaciona as características observadas com as dimensões do *output*, correspondentes aos valores de valência e estímulo.

Implementação

Os hiperparâmetros do modelo são novamente idênticos aos dos modelos anteriormente abordados, apresentando apenas a inclusão da dimensão oculta para a camada BiLSTM (hidden_dim).

A principal vantagem do modelo BiLSTM + Mecanismo de Atenção em relação ao modelo CNN + Mecanismo de Atenção é a capacidade que apresenta de captar dependências sequenciais e informação contextual nas letras através da camada BiLSTM, que processa a sequência introduzida em ambas as direções. Isto permite ao modelo considerar tanto o contexto das palavras passadas como futuras e assim captar dependências a longo prazo e compreender o contexto global da música.

No entanto, à semelhança dos modelos anteriores, também apresenta desvantagens relativamente à complexidade computacional, devido à utilização do modelo BERT, camada BiLSTM e mecanismo de atenção. E, tal como o modelo CNN + Mecanismo de Atenção, embora o mecanismo de atenção proporcione uma ligeira vantagem, o modelo BiLSTM + Mecanismo de Atenção continua a não ter o mesmo nível de interpretabilidade que os modelos mais simples apresentam. As interações complexas que ocorrem entre os BERT *embeddings*, a camada BiLSTM, o mecanismo de atenção e a camada totalmente conectada tornam extremamente difícil compreender o processo de decisão do modelo. Para além disso, também depende da qualidade e relevância do modelo BERT pré-treinado sendo que, quaisquer limitações ou parcialidades presentes nos dados podem repercutir-se na previsão de valores.

4.3.2.4 *Hierarchical Attention Network*

O modelo *Hierarchical Attention Network* (HAN), tal como os modelos abordados anteriormente, utiliza o poder do modelo BERT para compreensão contextual, e combina este com um mecanismo de atenção hierárquica para identificar representações ao nível das palavras e das frases.

A arquitetura do modelo HAN é a seguinte:

- Camada BERT: O último estado oculto do modelo BERT é utilizado como representação das palavras na letra.
- Camada de Atenção ao Nível da Palavra: É aplicado um mecanismo de atenção às representações ao nível das palavras, sendo os pesos de cada uma destas calculados utilizando uma camada linear (self.attention) seguida de uma função *softmax*. A representação ao nível da palavra é obtida através da soma dos estados ocultos em função dos pesos de atenção.

- Codificação ao Nível da Frase: As representações ao nível da palavra são passadas por uma camada *Gated Recurrent Unit* (GRU) que capta o contexto ao nível da frase.
- Previsão Final: A representação ao nível da frase passa por uma camada totalmente ligada (self.fc) que prevê então os valores de valência e estímulo.

O modelo HAN apresenta várias vantagens em relação aos modelos previamente abordados. Um dos seus benefícios é a capacidade de compreender a estrutura hierárquica das letras através da aplicação de mecanismos de atenção ao nível das palavras e das frases. Esta modelagem permite ao modelo considerar a importância de palavras individuais e simultaneamente, o contexto e a relação geral entre as frases. Este mecanismo de atenção incorporado permite ao modelo focar-se nas palavras mais relevantes para a previsão das emoções, enquanto a atenção ao nível das frases permite atribuir importância a diferentes frases com base no seu significado emocional.

Outra vantagem deste modelo, tal como os modelos anteriores, é a possibilidade de tirar partido das capacidades do modelo BERT para uma compreensão da informação contextual e das nuances emocionais da letra muito mais eficaz.

Comparativamente aos modelos LSTM e BiLSTM + Atenção, o modelo HAN oferece uma abordagem mais sofisticada para modelar as dependências sequenciais nas letras. Enquanto os modelos LSTM e BiLSTM processam os *tokens* sequencialmente, o modelo HAN incorpora uma estrutura hierárquica que considera tanto as representações ao nível da palavra como ao nível da frase. Isso permite ao modelo HAN capturar a progressão emocional e as dependências em diferentes granularidades, o que pode levar a previsões mais fiáveis.

As desvantagens do modelo HAN são semelhantes à dos seus parceiros de *Deep Learning*: complexidade computacional, sendo que requer um uso de recursos intensivo; dificuldade de interpretação devido ao uso do modelo BERT com o mecanismo de atenção hierárquica, que torna difícil atribuir a importância de partes específicas das letras às previsões finais e dependência da qualidade do modelo BERT pré-treinado.

4.3.3 Modelos de Fusão

O modelo de fusão é uma abordagem de aprendizagem conjunta (*ensemble learning*) que tem como objetivo aproveitar os pontos fortes de diferentes modelos de modo a melhorar o desempenho global de previsão. O modelo de fusão criado nesta investigação combina as previsões dos modelos com melhor desempenho nas modalidades exploradas anteriormente, sendo formado pelas previsões dos modelos *Random Forest* e CNN + Mecanismo de Atenção.

O modelo apresenta uma arquitetura composta por duas fases:

- Modelos Base:
    1. Modelo *Random Forest*: Treinado nos valores de valência e estímulo obtidos pelo modelo original *Random Forest*. Aprende a associar os valores previstos pelo modelo original com os valores nos quais foi treinado, disponibilizados na base de dados.
    2. Modelo de Regressão Linear: Treinado nos valores de valência e estímulo obtidos pelo modelo CNN + Mecanismo de Atenção. Tal como o modelo anterior, aprende a associar os valores previstos pelo modelo CNN + Mecanismo de Atenção com os valores nos quais este foi treinado, disponibilizados na base de dados.
- Meta- Modelo:
    1. Outro modelo *Random Forest* que utiliza as previsões dos modelos base como características de aprendizagem para efetuar as previsões finais. É importante referir que foram explorados outros meta-modelos, incluindo Regressão Linear e Rede Neural, no entanto, o meta-modelo *Random Forest* foi o que obteve os melhores resultados das três abordagens.

O processo de fusão inicia com o carregamento de dois ficheiros CSV: "random_forest_predictions.cs"' e "cnn_predictions.csv", que contêm as previsões obtidas pelos modelos *Random Forest* e CNN associadas aos valores originais disponibilizados na base de dados. Os dados utilizados como *input* são os valores previstos pelos modelos e as variáveis alvo são os valores originais disponíveis no *dataset*.

Os dados são divididos em conjuntos de treino e de teste utilizando a função "train_test_split". O conjunto de treino é utilizado para treinar os modelos de base e meta-modelo, enquanto o conjunto de teste é utilizado para avaliar o seu desempenho.

Depois de treinar os modelos, as previsões são feitas no conjunto de teste usando cada modelo separadamente. Os valores de valência previstos pelo modelo base *Random Forest* e os

valores de estímulo previstos pelo modelo base de Regressão Linear são combinados num novo *DataFrame* chamado "ensemble_predictions". O meta-modelo treina então com as previsões deste novo *DataFrame* como valores de *input* e usa os valores reais de valência e estímulo como variáveis-alvo.

4.4 Conclusão

Neste capítulo exploramos os diversos modelos utilizados na investigação para previsão emocional das métricas de valência e estímulo através de dados áudio e das letras de canções. Os modelos incluíram abordagens mais simples de *Machine Learning*, como Regressão Linear, *Random Forest*, *Gradient Boosting* e Máquinas de Vetores de Suporte (SVR), mas também arquiteturas mais complexas de *Deep Learning*, como CNN + Mecanismo de Atenção, LSTM, BiLSTM + Mecanismo de Atenção e ainda *Hierarchical Attention Network* (HAN). Também foi explorada a implementação do modelo de fusão, criado através das previsões dos modelos com melhor desempenho de cada modalidade.

Os modelos de *Machine Learning*, apesar de serem mais fáceis de interpretar e menos dispendiosos de um ponto de vista computacional, apresentam dificuldades em captar as relações complexas entre as características em estudo. A regressão linear pressupõe uma relação linear entre as características e as variáveis-alvo, o que pode nem sempre ser o caso. Os modelos *Random Forest* e *Gradient Boosting*, apesar de serem capazes de lidar com relações não lineares, são suscetíveis a sobreajuste e podem não generalizar corretamente a informação. Já o modelo SVR, embora eficaz no tratamento de relações não lineares, é mais dispendioso computacionalmente e requer uma afinação extensiva dos hiperparâmetros.

Já os modelos de *Deep Learning*, tiram partido do modelo BERT pré-treinado para recolha de informações contextuais e compreensão do significado semântico das letras. A utilização destes modelos reduz a necessidade de grandes quantidades de dados de treino rotulados, e os mecanismos de atenção que grande parte destes modelos apresentam (à exceção do modelo LSTM) permitem um foco superior nas partes mais relevantes das letras.

O modelo CNN com Mecanismo de Atenção combina as capacidades de extração de características das redes neurais com um mecanismo de atenção que permite concentrar nas sequências mais relevantes das letras. O modelo LSTM capta as dependências a longo prazo nas letras das músicas, processando sequencialmente os BERT *embeddings*. O modelo BiLSTM com mecanismo de atenção melhora a abordagem LSTM, considerando os contextos passado e futuro e incorpora um mecanismo de atenção para também concentrar nas palavras e frases mais

relevantes. O modelo HAN incorpora uma estrutura hierárquica que considera tanto as representações ao nível das palavras como ao nível das frases, permitindo uma captação da progressão emocional e das dependências em diferentes granularidades.

No entanto, os modelos *Deep Learning* também apresentam o seu próprio conjunto de desafios. São computacionalmente complexos, exigindo recursos e tempo de treino excessivos. As interações complexas entre os modelos BERT, os mecanismos de atenção e as camadas totalmente ligadas dificultam a interpretação do processo de tomada de decisão do modelo. Além disso, estes modelos dependem da qualidade e relevância do modelo BERT pré-treinado, e quaisquer tendências ou limitações presentes nos dados podem propagar-se para as previsões emocionais.

Para além dos modelos individuais, foi também explorado um modelo de fusão para aumento do desempenho da previsão e aproveitamento dos benefícios dos modelos com melhor desempenho. Este modelo combina as previsões do modelo *Random Forest*, treinado com características áudio, e o modelo CNN + Mecanismo de Atenção, treinado com as letras das músicas. No entanto, também introduz uma complexidade adicional e apresenta uma maior dificuldade na interpretabilidade. comparativamente aos modelos mais simples.

Concluindo, a escolha do modelo depende dos requisitos específicos da tarefa em causa. Se a interpretabilidade e a eficiência computacional forem as principais preocupações, são preferíveis modelos mais simples como Regressão Linear, *Random Forest* ou *Gradient* Boosting. No entanto, se o objetivo principal é a compreensão de relações complexas, informações contextuais e uma maior precisão das previsões, os modelos de *Deep Learning* são os mais adequados. A abordagem do modelo de fusão oferece uma abordagem interessante para combinar os pontos fortes de diferentes modelos, mas também apresenta as suas próprias desvantagens no que diz respeito à complexidade e à interpretabilidade.

Implementação

5. Conclusões e Trabalho Futuro

5.1 Satisfação dos Objetivos

O principal objetivo desta investigação foi o desenvolvimento e avaliação de diversos modelos multimodais para previsão do conteúdo emocional na música. De modo a atingir este objetivo, foram exploradas diversas arquiteturas de *Machine Learning*, como Regressão Linear, *Random Forest*, *Gradient* Boosting, SVR, *Deep Learning*: CNN + Mecanismo de Atenção, LSTM, BiLSTM + Mecanismo de Atenção e HAN e ainda modelos de fusão que unem as vantagens de ambas as modalidades.

Ao longo da investigação, foram implementados com sucesso os modelos referidos, sendo estes treinados no conjunto de dados WASABI, que contem informação áudio (ritmo, volume e acordes), mas também as letras das canções, tendo cada uma das entradas na base de dados valores únicos de valência e estímulo associados.

Esta investigação é constituída então por três componentes distintas: criação de modelos de *Machine Learning* para interpretação de dados áudio, criação de modelos de *Deep Learning* para interpretação de dados líricos e a criação de modelos de fusão para interpretação multimodal. As Tabelas 1, 2 e 3 apresentam os resultados obtidos para todos os modelos criados e treinados. A Tabela 4 compila os melhores resultados de cada uma das modalidades.

Tabela 1: Resultados dos modelos de *Machine Learning*: modadlidade áudio

Modelo	*Valência*			*Estímulo*		
	MAE	*MSE*	R^2	*MAE*	*MSE*	R^2
Regressão Linear	0.2682	0.1202	0.0109	0.1739	0.0563	0.0246
Random Forest	0.2652	0.1169	0.0442	0.1699	0.0548	0.0662
Gradient Boosting	0.2661	0.1178	0.0367	0.1705	0.0552	0.0589
SVR	0.2682	0.1209	0.0000	0.1784	0.0579	0.0000

Tabela 2: Resultados dos modelos de *Deep Learning*: modalidade lírica

Modelo	*Valência*			*Estímulo*		
	MAE	*MSE*	R^2	*MAE*	*MSE*	R^2
CNN + Mecanismo de Atenção	0.0883	0.0141	0.9196	0.0822	0.0122	0.8780
LSTM	0.3294	0.1664	0.0488	0.2377	0.0998	0.0026
BiLSTM + Mecanismo de Atenção	0.3213	0.1622	0.0732	0.2357	0.1002	0.0000
HAN	0.4102	0.2564	0.0000	0.2429	0.0987	0.0127

Tabela 3: Resultados dos modelos de fusão.

Meta-Modelo	*Valência*			*Estímulo*		
	MAE	*MSE*	R^2	*MAE*	*MSE*	R^2
Regressão Linear	0.0008	0.0001	0.7315	0.0020	0.0007	0.0361
Random Forest	0.0004	0.0001	0.8573	0.0005	0.0001	0.8387
Rede Neural	0.0014	0.0001	0.7464	0.0018	0.0007	0.0423

Tabela 4: Modelos com melhor desempenho para cada uma das modalidades.

Modelo	*Valência*			*Estímulo*		
	MAE	*MSE*	R^2	*MAE*	*MSE*	R^2
Random Forest	0.2652	0.1169	0.0442	0.1699	0.0548	0.0662
CNN + Mecanismo de Atenção	0.0883	0.0141	0.9196	0.0822	0.0122	0.8780
Modelo Fusão: Random Forest	0.0004	0.0001	0.8573	0.0005	0.0001	0.8387

De modo a compreender e comparar os resultados obtidos, foram selecionados três estudos da literatura que propõem abordagens semelhantes à desta investigação.

Delbouys et al. (2018) introduzem um mecanismo multimodal que combina um modelo CNN para estudo de características áudio, com um modelo CNN-LSTM treinado num conjunto de letras de música, sendo os resultados destes modelos concatenados e passados por camadas totalmente conectadas de modo a gerar as previsões. Este estudo, para além de utilizar uma arquitetura de reconhecimento emocional semelhante à que é abordada nesta investigação, também utiliza o *Million Song Dataset annotated by Deezer* (MSDD) (Bertin-Mahieux et al., 2011), que serviu como base para a criação do conjunto de dados utilizado nesta investigação, o *dataset* WASABI. O MSDD contém 18.644 canções anotadas com diferentes valores de valência e estímulo.

Pyrovolakis et al. (2021) propõe um modelo que utiliza um modelo BERT pré-treinado para o processamento de letras e um modelo CNN para o processamento de características áudio, sendo os resultados destes também posteriormente concatenados. Esta abordagem alcançou um melhor desempenho num conjunto de dados diferente, o MoodyLyrics (Çano & Morisio, 2017), que é constituído por 2.595 canções anotadas com uma de quatro etiquetas de humor (feliz, zangado, triste, relaxado), também baseado no modelo de Russel.

Zhao et al. (2022) propõe uma abordagem multimodal através da implementação de um único modelo, formado por três módulos distintos: um para extração de características de áudio, outro para extração de características de texto e por último, um módulo de fusão hierárquica de características multimodais. Este modelo foi treinado nos dois conjuntos de dados empregues pelas prévias investigações: o MSDD e o MoodyLyrics.

A Tabela 5 compara os resultados da literatura referidos com os melhores resultados desta investigação.

Tabela 5: Comparação dos resultados de diversas abordagens da literatura com os resultados da investigação presente.

Modelo	*Valência*			*Estímulo*		
	MAE	*MSE*	R^2	*MAE*	*MSE*	R^2
Random Forest	0.2652	0.1169	0.0442	0.1699	0.0548	0.0662
CNN + Mecanismo de Atenção	0.0883	0.0141	0.9196	0.0822	0.0122	0.8780
Modelo Fusão	0.0004	0.0001	0.8573	0.0005	0.0001	0.8387
Modelo de Delbouys et al. (2018)	-	-	0.2190	-	-	0.2320
Modelo de Pyrovolakis et al. (2021)	-	-	0.2580	-	-	0.2700
Modelo de Zhao et al. (2022)	-	-	0.3060	-	-	0.3110

O modelo *Random Forest* obteve um valor R^2 de 0,0442 para a previsão da valência e de 0,0662 para a previsão do estímulo. Estes valores relativamente baixos indicam que o modelo apresenta dificuldade em captar as relações complexas entre as características áudio e as dimensões emocionais. Para além disto, também apresenta valores de MAE e MSE mais elevados comparativamente às abordagens de *Deep Learning*, o que evidencia as suas limitações.

Em contrapartida, o modelo CNN + Mecanismo de Atenção demonstra um desempenho superior, com um valor R^2 de 0,9196 para a previsão da valência e de 0,8780 para a previsão do estímulo. Estes valores sugerem que o modelo capta efetivamente os padrões e as relações subjacentes entre as letras e as emoções, e os valores baixos para MAE e MSE (0,0883 e 0,0141 para valência, 0,0822 e 0,0122 para estímulo) reforçam a capacidade do modelo para previsão do conteúdo emocional.

Comparando os resultados obtidos com os da literatura, observa-se que os modelos propostos por Delbouys et al., Pyrovolakis et al. e Zhao et al. atingem valores de R^2 que variam entre 0,219 e 0,306 para a previsão de valência e 0,232 e 0,311 para a previsão de estímulo. Embora estes valores sejam superiores aos obtidos pelo modelo de *Random Forest*, são

significativamente inferiores aos valores obtidos pelos modelos de fusão e CNN + Mecanismo de Atenção da presente investigação.

O modelo de fusão demonstra então resultados notáveis quando comparado com os modelos da literatura, com um valor R^2 de 0,8573 para a previsão da valência e de 0,8387 para a previsão do estímulo. O seu desempenho ultrapassa por uma margem considerável os valores R^2 mais elevados na literatura, o que demonstra a sua capacidade de combinar os pontos fortes dos modelos selecionados e captar diferentes aspetos dos dados. Para além disto, os valores MAE e MSE que apresenta (0,0004 e 0,0001 para a valência, 0,0005 e 0,0001 para o estímulo) são também substancialmente baixos.

Estas diferenças de desempenho podem ser atribuídas a diversos fatores. Os conjuntos de dados utilizados para o treino e a arquitetura de cada um dos modelos. Delbouys et al, utilizam o *Million Song Dataset annotated by Deezer* que, sendo uma das várias bases de dados utilizada na criação do conjunto de dados empregue nesta investigação, apresenta uma fração da informação disponibilizada pelo *dataset* WASABI. Para além disto, os modelos também apresentam uma concatenação simples, que pode não ser suficiente para compreender as relações complexas entre as duas modalidades. Finalmente, o processamento sequencial das letras apresentado pela arquitetura CNN – LSTM, apesar de conseguir modelar dependências sequenciais, também pode apresentar dificuldades na captação de dependências a longo prazo, o que influencia a compreensão do contexto geral das letras.

Pyrovolakis et al. e Zhao et al. utilizam modelos BERT pré-treinados para o processamento de letras, semelhantes à abordagem desta investigação. No entanto, estes modelos diferem na componente de processamento de áudio e no mecanismo de fusão, sendo que utilizam um modelo de áudio baseado em CNN para concatenação direta dos resultados, enquanto Zhao et al. introduzem um módulo de Atenção Modal Cruzada (CMA) de modo a realizar a fusão das características. O mecanismo de atenção e as ligações residuais do módulo CMA podem possivelmente contribuir para o seu desempenho superior comparativamente aos outros dois estudos.

O desempenho superior do modelo CNN + Mecanismo de Atenção desta investigação pode ser justificado, primeiramente, pelo mecanismo de atenção que apresenta, que permite ao modelo concentrar nas partes mais relevantes da letra e realizar de uma forma eficaz a previsão emocional. Em segundo lugar, a sua arquitetura inclui camadas convolucionais para extração de características e camadas totalmente ligadas para previsão, o que permite ao modelo uma aprendizagem mais aprofundada dos padrões entre letras e emoções. Por último, os BERT *embeddings* pré-treinados fornecem uma base sólida e indispensável para a compreensão contextual das letras.

Relativamente ao desempenho do modelo de fusão, é de notar que não só é superior ao dos modelos da literatura, como também ultrapassa o desempenho individual do modelo CNN +

Mecanismo de Atenção. Isto sugere que a abordagem "*ensemble learning*" aproveita de uma forma eficaz a informação complementar fornecida pelas modalidades áudio e lírica, resultando numa melhor previsão das emoções. É de notar a ausência dos valores de MAE e MSE para os modelos da literatura, o que limita a extensão da sua análise e comparação.

Portanto, através da investigação realizada foi possível concluir que:

- Os modelos de *Deep Learning*, particularmente os que incorporam mecanismos de atenção, superam significativamente as abordagens tradicionais de *Machine Learning* para tarefas de reconhecimento emocional.
- Apesar dos resultados extremamente positivos das arquiteturas de *Deep Learning*, a implementação dos modelos de fusão demonstrou resultados superiores ao de que um dos modelos individuais de cada uma das modalidades.
- A incorporação de características áudio e lírica conduz a melhorias drásticas no desempenho dos modelos para reconhecimento emocional.

No entanto, apesar dos resultados positivos, é também importante reconhecer as limitações do estudo:

1. A perceção emocional é inerentemente subjetiva e varia por inúmeras razões, sendo difícil de constatar verdades factuais aquando do estudo deste tópico.
2. Embora os escolhidos utilizados para treino dos modelos sejam pertinentes e relevantes no estudo da área de MER, podem não ser apresentar capacidade suficiente de englobar totalmente a complexidade da perceção emocional.
3. Embora os modelos *Deep Learning* e, consequentemente, os de fusão, tenham apresentado um desempenho excecional, o seu processo de tomada de decisão é extremamente difícil de interpretar, o que limita a capacidade de compreensão de como estes obtêm as suas previsões e quais os fatores que mais contribuem nestas previsões.

5.2 Trabalho Futuro

Embora o modelo CNN + Mecanismo de Atenção e o modelo de fusão tenham demonstrado um desempenho extremamente positivo na previsão emocional, existem diversas perspetivas que podem ser abordadas para melhorias em futuras investigações.

O estudo dos modelos de *Deep Learning* foca-se apenas na modalidade da letra para previsão do conteúdo emocional. Seria interessante, para além de utilizarem esta modalidade, efetuar o processo de treino na informação áudio aplicada aos modelos de *Machine Learning*.

As características em estudo podem, para além de ser as que são disponibilizadas na base de dados WASABI, também ser extraídas diretamente do áudio, obtendo assim novas características e uma informação mais detalhada da música. Características relevantes para esta finalidade incluem coeficientes cepstrum de frequência mel (*Mel-frequency cepstral coefficients*: MFCC), cromagramas e espectrogramas.

Todos os modelos da presente investigação foram treinados no conjunto de dados WASABI, que fornece uma quantidade vasta de informação musical. No entanto, foi utilizada apenas uma parte desta informação, sendo de grande interesse explorar outras das características disponibilizadas, tais como pormenores sobre o artista, discografia ou até ano de produção. A incorporação de informações específicas sobre o artista, como género, estilo ou contexto histórico, podem fornecer características interessantes para o contexto das letras. Do mesmo modo, considerar o ano de produção pode também ajudar a captar tendências temporais e mudanças no conteúdo das letras ao longo do tempo.

Uma das principais desvantagens da utilização de modelos de *Deep Learning* é a sua fraca interpretabilidade. É difícil compreender como é que o modelo chega às previsões que faz e quais as características que mais contribuem para esta classificação. Futuras investigações podem centrar-se no desenvolvimento de técnicas que visam a facilitar esta interpretabilidade, através de técnicas como visualização dos pesos de atenção, mapas de saliência ou análise da importância das características. Isto poderia fornecer informações sobre o processo de decisão do modelo e assim ajudar a identificar potenciais erros ou enviesamentos.

É, também interessante explorar o potencial dos modelos de fusão que, nesta investigação, ultrapassaram os modelos individuais e demonstraram capacidades notáveis. Podem ser investigadas diferentes estratégias de fusão, como o cálculo da média ponderada ou outras técnicas mais avançadas, de modo a melhorar o desempenho da previsão. Tal como nas abordagens *Deep Learning*, também é interessante aprofundar o tópico da interpretabilidade dos modelos através do desenvolvimento de técnicas que permitam compreender os contributos de cada um dos modelos para a tomada de decisões. Além disso, a integração de outras modalidades poderá também proporcionar uma compreensão mais abrangente do conteúdo emocional.

Ao considerar estas diferentes abordagens para futuras investigações: a integração de características de áudio, a exploração de arquiteturas de modelos alternativos, o desenvolvimento de técnicas para melhoria da interpretabilidade, o aproveitamento da vasta informação disponível no conjunto de dados WASABI e o desenvolvimento de modelos de fusão, é possível continuar a melhorar o desempenho dos modelos para previsão do conteúdo emocional e assim desenvolver um sistema mais abrangente e fiável.

Referências

(Yang *et al.*, 2017) Yang, X., Dong, Y., & Li, J. (2017). Review of data features-based music emotion recognition methods. *Multimedia Systems*, *24*(4), 365–389. https://doi.org/10.1007/s00530-017-0559-4

(He *et al.,* 2020) He, Z., Li, Z., Yang, F., Wang, L., Li, J., Zhou, C., & Pan, J. (2020). Advances in multimodal emotion recognition based on Brain–Computer interfaces. *Brain Sciences*, *10*(10), 687. https://doi.org/10.3390/brainsci10100687

(Cheng *et al.*, 2017) Cheng, Z., Shen, J., Zhu, L., Kankanhalli, M. S., & Nie, L. (2017). Exploiting Music Play Sequence for Music Recommendation. *IJCAI*, 3654-3660.

(Shen *et al.*, 2017) Shen, J., Cheng, Z., Nie, L., Chua, T. S., & Kankanhalli, M. (2017). Exploring user-specific information in music retrieval. *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, 655-664.

(Gunawan *et al.*, 2020) Gunawan, A. a. S., Iman, A. P., & Suhartono, D. (2020). Automatic music generator using recurrent neural network. *International Journal of Computational Intelligence Systems*, *13*(1), 645. https://doi.org/10.2991/ijcis.d.200519.001

(De Witte *et al.*, 2020) De Witte, M., Da Silva Pinho, A., Stams, G. J., Moonen, X., Bos, A. E. R., & Van Hooren, S. (2020). Music therapy for stress reduction: a systematic review and meta-analysis. *Health Psychology Review*, *16*(1), 134–159. https://doi.org/10.1080/17437199.2020.1846580

(Schedl *et al.*, 2018) Schedl, M., Zamani, H., Chen, C., Deldjoo, Y., & Elahi, M. (2018). Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, *7*(2), 95–116. https://doi.org/10.1007/s13735-018-0154-2

(Hevner, 1936) Hevner, K. (1936). Experimental studies of the elements of expression in music. *The American Journal of Psychology*, *48*(2), 246–268. https://doi.org/10.2307/1415746https://www.jstor.org/stable/1415746

(Russell, 1980) Russell, J. A. (1980). A circumplex model of affect. *Journal of Personality and Social Psychology*, *39*(6), 1161–1178. https://doi.org/10.1037/h0077714

(Parisi *et al.*, 2019) Parisi, L., Francia, S., Olivastri, S., & Tavella, M. S. (2019). *Exploiting synchronized lyrics and vocal features for music emotion detection*. https://arxiv.org/abs/1901.04831

(Hu et al., 2008) Hu, X., Downie, J. S., Laurier, C., & Ehmann, A. F. (2008). The 2007 MIREX audio mood classification task: Lessons learned. *ISMIR*, 462-467.

(Kim *et al.* 2010) Kim, Y. E., Schmidt, E. M., Migneco, R., Morton, B. G., Richardson, P., Scott, J., & Turnbull, D. (2010). Music emotion recognition: A state of the art review. *ISMIR*, 937-952.

(Hu *et al.*, 2009) Hu, X., Downie, J. S., & Ehmann, A. F. (2009). Lyric text mining in music mood classification. *American Music*, 2-209. https://www.researchgate.net/publication/220723046_Lyric_Text_Mining_in_Music_Mood_Classification

(Laurier *et al.*, 2008) Laurier, C., Grivolla, J., & Herrera, P. (2008). Multimodal Music Mood Classification Using Audio and Lyrics. *2008 Seventh International Conference on Machine Learning and Applications*, 688-693. https://doi.org/10.1109/icmla.2008.96

(Downie *et al.*, 2008) Downie, X. H. J. S., Laurier, C., & Ehmann, M. B. A. F. (2008). The 2007 MIREX audio mood classification task: Lessons learned. *Proc. 9th Int. Conf. Music Inf. Retrieval*, 462-467.

(Alajanki *et al.*, 2016) Alajanki, A., Yang, Y. H., & Soleymani, M. (2016). Benchmarking music emotion recognition systems. *PloS one*, 835-838.

(Korzeniowski *et al.*, 2020) Korzeniowski, F., Nieto, O., McCallum, M., Won, M., Oramas, S., & Schmidt, E. (2020). *Mood classification using listening data*. https://arxiv.org/abs/2010.11512

(Fell *et al.*, 2022) Fell, M., Cabrio, E., Tikat, M., Michel, F., Buffa, M., & Gandon, F. (2022). The WASABI song corpus and knowledge graph for music lyrics analysis. *Language Resources and Evaluation*, *57*(1), 89–119. https://doi.org/10.1007/s10579-022-09601-8

(Meseguer-Brocal *et al.*, 2017) Meseguer-Brocal, G., Peeters, G., Pellerin, G., Buffa, M., Cabrio, E., Faron Zucker, C., Giboin, A., Mirbel, I., Hennequin, R., Moussallam, M. (2017). *WASABI: a Two Million Song Database Project with Audio and Cultural Metadata plus WebAudio enhanced Client Applications*. https://qmro.qmul.ac.uk/xmlui/handle/123456789/26123

(Tzanetakis & Cook, 2000) Tzanetakis, G., & Cook, P. R. (2000). MARSYAS: a framework for audio analysis. *Organised Sound*, *4*(3), 169–175. https://doi.org/10.1017/s1355771800003071

(Mathieu *et al.*, 2010) Mathieu, B., Essid, S., Fillon, T., Prado, J., & Richard, G. (2010). YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software. *ISMIR 2010*, 441-446.

Referências

(Lartillot *et al.*, 2007) Lartillot, O., & Toiviainen, P. (2007). A Matlab toolbox for musical feature extraction from audio. *International conference on digital audio effects*, 244.

(McKay *et al.*, 2005) McKay, C., Fujinaga, I., & Depalle, P. (2005). jAudio: A feature extraction library. In *Proceedings of the international conference on music information retrieval*, 600-3.

(Han *et al.*, 2014) Han, W. J., Li, H. F., Ruan, H. B., & Ma, L. (2014). Review on speech emotion recognition. *Journal of software*, *25*(1), 37-50.

(Liu *et al.*, 2017) Liu, X., Chen, Q., Wu, X., & Liu, Y. (2017). CNN based music emotion classification. https://www.researchgate.net/publication/316270219_CNN_based_music_emotion_classification

(Chen *et al.*, 2016) Chen, P., Zhao, L., Xin, Z., Qiang, Y., Zhang, M., & Li, T. (2016, December). A scheme of MIDI music emotion classification based on fuzzy theme extraction and neural network. In *2016 12th International Conference on Computational Intelligence and Security (CIS)* (pp. 323-326). IEEE.

(Juslin & Laukka, 2004) Juslin, P., & Laukka, P. (2004). Expression, Perception, and Induction of Musical Emotions: A review and a questionnaire study of everyday listening. *Journal of New Music Research*, *33*(3), 217–238. https://doi.org/10.1080/0929821042000317813

(Yang *et al.*, 2004) Yang, D., & Lee, W. S. (2004). Disambiguating Music Emotion Using Software Agents. *ISMIR 2004*, 218-223.

(He *et al.*, 2008) He, H., Jin, J., Xiong, Y., Chen, B., Sun, W., & Zhao, L. (2008). Language Feature Mining for Music Emotion Classification via Supervised Learning from Lyrics. *Lecture Notes in Computer Science*, 426–435. https://doi.org/10.1007/978-3-540-92137-0_47

(Hu *et al.*, 2009) Hu, X., Downie, J. S., & Ehmann, A. F. (2009). Lyric text mining in music mood classification. *American music*, *183*, 2-209.

Referências

(Malheiro *et al.*, 2018) Malheiro, R., Panda, R., Gomes, P. J. S., & Paiva, R. P. (2018). Emotionally-Relevant features for classification and regression of music lyrics. *IEEE Transactions on Affective Computing*, *9*(2), 240–254. https://doi.org/10.1109/taffc.2016.2598569

(Devlin *et al.*, 2018) Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 4171–4186.

(Müller *et al.*, 2023) Müller, M., Salathé, M., & Kummervold, P. E. (2023). COVID-Twitter-BERT: A natural language processing model to analyse COVID-19 content on Twitter. *Frontiers in Artificial Intelligence*, *6*. https://doi.org/10.3389/frai.2023.1023281

(Perez *et al.*, 2022) Perez, I., & Reinauer, R. (2022). The topological BERT: Transforming attention into topology for natural language processing. *arXiv preprint arXiv:2206.15195*.

(Thammasan *et al.*, 2016) Thammasan, N., Fukui, K., & Numao, M. (2016). Application of deep belief networks in eeg-based dynamic music-emotion recognition. *2016 International Joint Conference on Neural Networks (IJCNN)*. https://doi.org/10.1109/ijcnn.2016.7727292~

(Hu *et al.*, 2018) Hu, X., Li, F., & Ng, T. D. J. (2018, September). On the Relationships between Music-induced Emotion and Physiological Signals. *ISMIR 2018*, 362-369.

(Nawa *et al.*, 2018) Nawa, N. E., Callan, D. E., Mokhtari, P., Ando, H., & Iversen, J. R. (2018). Decoding music-induced experienced emotions using functional magnetic resonance imaging - Preliminary results. *2018 International Joint Conference on Neural Networks (IJCNN)*. https://doi.org/10.1109/ijcnn.2018.8489752

Referências

(Hinton *et al.*, 2006) Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

(Soleymani *et al.*, 2013) Soleymani, M., Micheal, N., Schmidt, E. K., Sha, C., & Yang, Y. (2013). 1000 songs for emotional analysis of music. *Proceedings of the 2nd ACM International Workshop on Crowdsourcing for Multimedia*. https://doi.org/10.1145/2506364.2506365

(Aljanaki *et al.*, 2017) Aljanaki, A., Yang, Y., & Soleymani, M. (2017). Developing a benchmark for emotional analysis of music. *PLOS ONE*, *12*(3), e0173392. https://doi.org/10.1371/journal.pone.0173392

(Speck *et al.*, 2011) Speck, J. A., Schmidt, E. M., Morton, B. G., & Kim, Y. E. (2011). A Comparative Study of Collaborative vs. Traditional Musical Mood Annotation. *ISMIR 2011*, 549-554.

(Turnbull *et al.*, 2007) Turnbull, D., Barrington, L., Torres, D., & Lanckriet, G. R. G. (2007). Towards musical query-by-semantic-description using the CAL500 data set. *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. https://doi.org/10.1145/1277741.1277817

(Wang *et al.*, 2014) Wang, S. Y., Wang, J. C., Yang, Y. H., & Wang, H. M. (2014). Towards time-varying music auto-tagging based on cal500 expansion. *2014 IEEE International Conference on Multimedia and Expo (ICME)*, 1-6.

(Zentner *et al.*, 2008) Zentner, M., Grandjean, D., & Scherer, K. R. (2008). Emotions evoked by the sound of music: Characterization, classification, and measurement. *Emotion*, *8*(4), 494–521. https://doi.org/10.1037/1528-3542.8.4.494

Referências

(Keelawat *et al.*, 2019) Keelawat, P., Thammasan, N., Kijsirikul, B., & Numao, M. (2019). Subject-Independent Emotion Recognition During Music Listening Based on EEG Using Deep Convolutional Neural Networks. *2019 IEEE 15th International Colloquium on Signal Processing & Its Applications (CSPA)*. https://doi.org/10.1109/cspa.2019.8696054

(Yang *et al.*, 2020) Yang, P., Kuang, S., Wu, C., & Hsu, J. (2020). Predicting music emotion by using convolutional neural network. *Lecture Notes in Computer Science*, 266–275. https://doi.org/10.1007/978-3-030-50341-3_21

(Ma *et al.*, 2017) Ma, Y., Li, X., Xu, M., Jia, J., & Cai, L. (2017). Multi-scale Context Based Attention for Dynamic Music Emotion Prediction. *Proceedings of the 25th ACM International Conference on Multimedia*. https://doi.org/10.1145/3123266.3123408

(Liu *et al.*, 2019) Liu, H., Fang, Y., & Huang, Q. (2019). Music Emotion Recognition Using a Variant of Recurrent Neural Network. *2018 International Conference on Mathematics, Modeling, Simulation and Statistics Application*. https://doi.org/10.2991/mmssa-18.2019.4

(Dong *et al.*, 2019) Dong, Y., Yang, X., Zhao, X., & Li, J. (2019). Bidirectional Convolutional Recurrent Sparse Network (BCRSN): an efficient model for music emotion recognition. *IEEE Transactions on Multimedia*, *21*(12), 3150–3163. https://doi.org/10.1109/tmm.2019.2918739

(Li *et al.*, 2016) Li, X., Tian, J., Xu, M., Ning, Y., & Cai, L. (2016). DBLSTM-based multi-scale fusion for dynamic emotion prediction in music. *2016 IEEE International Conference on Multimedia and Expo (ICME)*. https://doi.org/10.1109/icme.2016.7552956

(Chaki *et al.*, 2020) Chaki, S., Doshi, P., Patnaik, P., & Bhattacharya, S. (2020). Attentive RNNs for Continuous-time Emotion Prediction in Music Clips. *AffCon@ AAAI*, 36-46.

(Weninger *et al.*, 2014) Weninger, F., Eyben, F., & Schuller, B. (2014). On-line continuous-time music mood regression with deep recurrent neural networks. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. https://doi.org/10.1109/icassp.2014.6854637

(Panda *et al.*, 2019) Panda, R. E. S. (2019). *Emotion-based analysis and classification of audio music*. Doctoral dissertation, Universidade de Coimbra.

(Fell et al., 2018) Fell, M., Nechaev, Y., Cabrio, E., & Gandon, F. (2018). Lyrics Segmentation: Textual Macrostructure Detection using Convolutions. ACL Anthology. https://aclanthology.org/C18-1174/

(Fell et al., 2019) Fell, M., Cabrio, E., Korfed, E., Buffa, M., & Gandon, F. (2019). Love Me, Love Me, Say (and Write!) that You Love Me: Enriching the WASABI Song Corpus with Lyrics Annotations. arXiv.org. https://arxiv.org/abs/1912.02477

(Fell, 2020) Fell, M. (2020). Natural language processing for music information retrieval: deep analysis of lyrics structure and content. https://theses.hal.science/tel-02587910/

(Fell, Cabrio, Corazza, et al., 2019) Fell, M., Cabrio, E., Corazza, M., & Gandon, F. (2019). Comparing automated methods to detect explicit content in song lyrics. https://hal.science/hal-02281137/

(Fell, Cabrio, Gandon, et al., 2019) Fell, M., Cabrio, E., Gandon, F., & Giboin, A. (2019). Song Lyrics Summarization Inspired by Audio Thumbnailing. Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019). https://doi.org/10.26615/978-954-452-056-4_038

(Delbouys et al., 2018) Delbouys, R., Hennequin, R., Piccoli, F., Royo-Letelier, J., & Moussallam, M. (2018, September 19). Music mood detection based on audio and lyrics with deep neural net. arXiv.org. https://arxiv.org/abs/1809.07276

(Bertin-Mahieux et al., 2011) Bertin-Mahieux, T., Ellis, D. P. W., Whitman, B., & Lamere, P. (2011). THE MILLION SONG DATASET. ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, 2011, Miami, Florida, 591–596. https://doi.org/10.7916/d8nz8j07

Referências

(Çano & Morisio, 2017) Çano, E., & Morisio, M. (2017). MoodyLyrics. ISMSI ’17: Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence. https://doi.org/10.1145/3059336.3059340

(Zhao et al., 2022) Zhao, J., Ru, G., Yu, Y., Wu, Y., Li, D., & Li, W. (2022). Multimodal music emotion recognition with hierarchical cross-modal attention network. In 2022 IEEE International Conference on Multimedia and Expo (ICME) (pp. 1-6). IEEE. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9859812

Referências

Anexos

A.1 Pré-processamento da base de dados

Script "Output ALL Entries of Song JSON.py", escrito na linguagem Python, que utiliza as bibliotecas re, json e ijson.

```
import re
import json
import ijson

# Specify the input file path
input_file = 'song.json'

# Specify the output file path for all entries
output_file_all = 'song_extracted_all_entries.json'

# Specify the output file path for the first 20000 entries
output_file_20000 = 'song_extracted_first_20000_entries.json'

# Fields to extract
fields_to_extract = [
    "_id",
    "lyrics",
    "bpm",
    "gain",
    "length",
    "chords_metadata",
    "arousal",
    "arousal_predicted",
    "valence",
    "valence_predicted"
]
```

```
def extract_entry(entry):
    entry_dict = {}
    # Extract the desired fields using regular expressions
    for field in fields_to_extract:
        if field == "_id":
            if "_id" in entry and "$oid" in entry["_id"]:
                entry_dict[field] = entry["_id"]["$oid"]
            else:
                entry_dict[field] = "{}"
        elif field == "lyrics":
            if "lyrics" in entry:
                lyrics = entry["lyrics"]
                # Remove HTML line breaks
                lyrics = re.sub(r'<br>', ' ', lyrics)
                # Replace HTML entity &apos; with apostrophe
                lyrics = lyrics.replace('&apos;', "'")
                entry_dict[field] = lyrics.strip()
            else:
                entry_dict[field] = ""
        elif field == "chords_metadata":
            if "chords_metadata" in entry:
                chords_metadata = entry["chords_metadata"]
                if "chordSequence" in chords_metadata:
                    chord_sequence = chords_metadata["chordSequence"]
                    for chord in chord_sequence:
                        chord["start"] = float(chord["start"])
                        chord["end"] = float(chord["end"])
                if "confidence" in chords_metadata:
                    chords_metadata["confidence"] =
float(chords_metadata["confidence"])
                if "duration" in chords_metadata:
                    chords_metadata["duration"] =
float(chords_metadata["duration"])
                entry_dict[field] = chords_metadata
            else:
                entry_dict[field] = {}
        else:
            if field in entry:
                entry_dict[field] = entry[field]
            else:
                entry_dict[field] = ""

    return entry_dict

# Process the input file
with open(input_file, 'r', encoding='utf-8') as file, \
     open(output_file_20000, 'w', encoding='utf-8') as file_20000, \
     open(output_file_all, 'w', encoding='utf-8') as file_all:
```

```
    entries = ijson.items(file, 'item')
    entry_count = 0
    for entry in entries:
        try:
            entry_dict = extract_entry(entry)
            json.dump(entry_dict, file_all)
            file_all.write('\n')

            if entry_count < 20000:
                json.dump(entry_dict, file_20000)
                file_20000.write('\n')

            entry_count += 1

            # Print progress every 1000 entries
            if entry_count % 1000 == 0:
                print(f"Processed {entry_count} entries.")
        except Exception as e:
            print(f"Error processing entry {entry_count}: {str(e)}")
            print(f"Skipping entry {entry_count} and continuing with the
next entry.")

print(f"Extracted data from the first 20000 entries written to
{output_file_20000}")
print(f"Extracted data from all entries written to {output_file_all}")
```

A.2 Modelos de *Machine Learning*

Regressão Linear

Script "Linear Regression SONG DATASET.py" escrito na linguagem Python, que utiliza as bibliotecas pandas, json, sklearn e matplotlib.

```
import pandas as pd
print("Pandas imported successfully.")

import json
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import matplotlib.pyplot as plt
```

```
print("Other libraries imported successfully.")

def process_batch(batch):
    print(f"Processing batch of size {len(batch)}...")

    # Function to parse the 'chords_metadata' column and extract features
    def extract_chords_features(chords_metadata):
        try:
            num_chords = len(chords_metadata['chordSequence'])
            avg_chord_duration = chords_metadata['duration'] / num_chords
            return pd.Series([num_chords, avg_chord_duration])
        except (KeyError, TypeError):
            return pd.Series([0, 0])

    # Extract features from the 'chords_metadata' column
    chords_features =
batch['chords_metadata'].apply(extract_chords_features)
    chords_features.columns = ['num_chords', 'avg_chord_duration']

    # Concatenate the extracted features with the batch
    batch = pd.concat([batch, chords_features], axis=1)

    # Convert numeric columns to appropriate data types
    numeric_columns = ['bpm', 'gain', 'length', 'arousal_predicted',
'valence_predicted']
    batch[numeric_columns] = batch[numeric_columns].apply(pd.to_numeric,
errors='coerce')

    print("Batch processed successfully.")
    return batch

# Initialize an empty list to store the processed batches
processed_batches = []

# Process the dataset in batches
batch_size = 1000
with open('song_extracted_all_entries.json', 'r') as file:
    print("Reading dataset file...")
    batch = []
    for line in file:
        song = json.loads(line)
        batch.append(song)
        if len(batch) == batch_size:
            print(f"Processing batch {len(processed_batches) + 1}...")
            batch_df = pd.DataFrame(batch)
            processed_batch = process_batch(batch_df)
            processed_batches.append(processed_batch)
            batch = []
```

```
            print(f"Batch {len(processed_batches)} processed
successfully.")

    # Process the remaining songs if any
    if batch:
        print("Processing remaining songs...")
        batch_df = pd.DataFrame(batch)
        processed_batch = process_batch(batch_df)
        processed_batches.append(processed_batch)
        print("Remaining songs processed successfully.")

print("All batches processed successfully.")

# Concatenate the processed batches into a single DataFrame
print("Concatenating processed batches...")
songs_df = pd.concat(processed_batches, ignore_index=True)
print("Concatenation completed.")

# Select the relevant features and target variables from the songs
dataset
features = ['bpm', 'gain', 'length', 'num_chords', 'avg_chord_duration']
target_valence = 'valence_predicted'
target_arousal = 'arousal_predicted'

# Extract the features and target variables
print("Extracting features and target variables...")
X = songs_df[features]
y_valence = songs_df[target_valence]
y_arousal = songs_df[target_arousal]
print("Features and target variables extracted successfully.")

# Split the data into training and testing sets
print("Splitting data into training and testing sets...")
X_train, X_test, y_train_valence, y_test_valence, y_train_arousal,
y_test_arousal, id_train, id_test = train_test_split(
    X, y_valence, y_arousal, songs_df['_id'], test_size=0.2,
random_state=42)
print("Data split completed.")

# Drop samples with missing values in the training set
print("Dropping samples with missing values in the training set...")
X_train_cleaned = X_train.dropna()
y_train_valence_cleaned = y_train_valence[X_train.notna().all(axis=1)]
y_train_arousal_cleaned = y_train_arousal[X_train.notna().all(axis=1)]
print("Samples with missing values dropped from the training set.")

# Create and train the linear regression models
print("Creating and training linear regression models...")
model_valence = LinearRegression()
```

```
model_arousal = LinearRegression()

print("Training valence model...")
model_valence.fit(X_train_cleaned, y_train_valence_cleaned)
print("Valence model training completed.")

print("Training arousal model...")
model_arousal.fit(X_train_cleaned, y_train_arousal_cleaned)
print("Arousal model training completed.")

# Drop samples with missing values in the test set
print("Dropping samples with missing values in the test set...")
X_test_cleaned = X_test.dropna()
y_test_valence_cleaned = y_test_valence[X_test.notna().all(axis=1)]
y_test_arousal_cleaned = y_test_arousal[X_test.notna().all(axis=1)]
id_test_cleaned = id_test[X_test.notna().all(axis=1)]
print("Samples with missing values dropped from the test set.")

# Make predictions on the cleaned test set
print("Making predictions on the cleaned test set...")
valence_pred = model_valence.predict(X_test_cleaned)
arousal_pred = model_arousal.predict(X_test_cleaned)
print("Predictions completed.")

# Compute evaluation metrics for valence (predicted)
print("Computing evaluation metrics for valence (predicted)...")
valence_pred_mae = mean_absolute_error(y_test_valence_cleaned,
valence_pred)
valence_pred_mse = mean_squared_error(y_test_valence_cleaned,
valence_pred)
valence_pred_r2 = r2_score(y_test_valence_cleaned, valence_pred)
print("Valence evaluation metrics computed.")

# Compute evaluation metrics for arousal (predicted)
print("Computing evaluation metrics for arousal (predicted)...")
arousal_pred_mae = mean_absolute_error(y_test_arousal_cleaned,
arousal_pred)
arousal_pred_mse = mean_squared_error(y_test_arousal_cleaned,
arousal_pred)
arousal_pred_r2 = r2_score(y_test_arousal_cleaned, arousal_pred)
print("Arousal evaluation metrics computed.")

# Create a DataFrame to store the original and predicted values, along
with the '_id'
print("Creating results DataFrame...")
results = pd.DataFrame({
    '_id': id_test_cleaned,
    'Valence_Predicted': y_test_valence_cleaned,
    'Valence_Pred_Model': valence_pred,
```

```
    'Arousal_Predicted': y_test_arousal_cleaned,
    'Arousal_Pred_Model': arousal_pred
})
print("Results DataFrame created.")

# Print the results
print("\nValence Metrics (Predicted):")
print("MAE:", valence_pred_mae)
print("MSE:", valence_pred_mse)
print("R-squared:", valence_pred_r2)

print("\nArousal Metrics (Predicted):")
print("MAE:", arousal_pred_mae)
print("MSE:", arousal_pred_mse)
print("R-squared:", arousal_pred_r2)

# Save the results to a JSON file
print("Saving results to JSON file...")
results.to_json('results_linear_regression.json', orient='records')
print("Results saved successfully.")

# Create scatter plots
print("Creating scatter plots...")
plt.figure(figsize=(8, 4))

# Scatter plot for valence (predicted)
plt.subplot(1, 2, 1)
plt.scatter(results['Valence_Predicted'], results['Valence_Pred_Model'],
alpha=0.5)
plt.xlabel('Predicted Valence (Gradient Boosting)')
plt.ylabel('Predicted Valence (Model)')
plt.title('Valence (Predicted)')

# Scatter plot for arousal (predicted)
plt.subplot(1, 2, 2)
plt.scatter(results['Arousal_Predicted'], results['Arousal_Pred_Model'],
alpha=0.5)
plt.xlabel('Predicted Arousal (Gradient Boosting)')
plt.ylabel('Predicted Arousal (Model)')
plt.title('Arousal (Predicted)')

plt.tight_layout()
plt.show()

print("Script execution completed.")
```

Random Forest

Script “Random Forest SONG DATASET.py” escrito na linguagem Python, que utiliza as bibliotecas pandas, json, pandas, numpy, sklearn e matplotlib.

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import matplotlib.pyplot as plt

# Create a mapping of chord labels to integer values
chord_mapping = {}

def extract_chords_metadata(chunk):
    def extract_num_chords(x):
        if isinstance(x, dict) and 'chordSequence' in x:
            return len(x['chordSequence'])
        else:
            return 0

    def extract_avg_chord_duration(x):
        if isinstance(x, dict) and 'chordSequence' in x and 'duration' in
x and len(x['chordSequence']) > 0:
            return x['duration'] / len(x['chordSequence'])
        else:
            return 0

    def extract_chord_labels(x):
        global chord_mapping
        if isinstance(x, dict) and 'chordSequence' in x:
            labels = [chord['label'] for chord in x['chordSequence'] if
'label' in chord]
            for label in labels:
                if label not in chord_mapping:
                    chord_mapping[label] = len(chord_mapping)
            return [chord_mapping[label] for label in labels]
        else:
            return []

    chunk['num_chords'] =
chunk['chords_metadata'].apply(extract_num_chords)
```

```
    chunk['avg_chord_duration'] =
chunk['chords_metadata'].apply(extract_avg_chord_duration)
    chunk['chord_labels'] =
chunk['chords_metadata'].apply(extract_chord_labels)
    return chunk

# Load the song_extracted_all_entries.json file in batches
batch_size = 1000
train_data = []
test_data = []
total_entries = 0
training_entries = 0

with pd.read_json('song_extracted_all_entries.json', lines=True,
chunksize=batch_size) as reader:
    for i, chunk in enumerate(reader):
        # Check if the required columns are present in the DataFrame
        required_columns = ['_id', 'bpm', 'gain', 'length',
'arousal_predicted', 'valence_predicted', 'chords_metadata']
        missing_columns = [col for col in required_columns if col not in
chunk.columns]

        if missing_columns:
            raise ValueError(f"The following required columns are missing
in the DataFrame: {missing_columns}")

        # Extract chords metadata
        chunk = extract_chords_metadata(chunk)

        # Replace empty string values with np.nan for relevant columns
        chunk['bpm'] = pd.to_numeric(chunk['bpm'], errors='coerce')
        chunk['gain'] = pd.to_numeric(chunk['gain'], errors='coerce')
        chunk['length'] = pd.to_numeric(chunk['length'], errors='coerce')
        chunk['arousal_predicted'] =
chunk['arousal_predicted'].apply(lambda x: np.nan if x == '' else x)
        chunk['valence_predicted'] =
chunk['valence_predicted'].apply(lambda x: np.nan if x == '' else x)

        # Select only the relevant columns
        chunk = chunk[['_id', 'bpm', 'gain', 'length',
'arousal_predicted', 'valence_predicted', 'num_chords',
'avg_chord_duration', 'chord_labels']]

        # Filter rows with missing or zero values
        chunk = chunk[(chunk['bpm'].notnull()) &
(chunk['gain'].notnull()) & (chunk['length'].notnull()) &
(chunk['arousal_predicted'].notnull()) &
(chunk['valence_predicted'].notnull()) & (chunk['num_chords'] != 0) &
(chunk['avg_chord_duration'] != 0)]
```

```
        if len(chunk) >= 5:  # Adjust the threshold as needed
            # Split the current batch into train and test sets
            train_chunk, test_chunk = train_test_split(chunk,
test_size=0.2, random_state=42)

            # Append the train and test data to the respective lists
            train_data.append(train_chunk)
            test_data.append(test_chunk)

            training_entries += len(train_chunk)
        else:
            # If the number of samples is below the threshold, append the
entire chunk to the train data
            train_data.append(chunk)
            training_entries += len(chunk)

        total_entries += len(chunk)
        print(f"Loaded {total_entries} entries")

        # Print the last 5 rows of the chunk
        print(chunk.tail())

        # Print the column names
        print(chunk.columns)

print(f"Data loading completed. Total entries: {total_entries}, Training
entries: {training_entries}")

# Concatenate the train and test data
train_df = pd.concat(train_data, ignore_index=True)
test_df = pd.concat(test_data, ignore_index=True)

print("Train data shape:", train_df.shape)
print("Test data shape:", test_df.shape)

# Convert target variables to float
train_df['valence_predicted'] =
train_df['valence_predicted'].astype(float)
train_df['arousal_predicted'] =
train_df['arousal_predicted'].astype(float)
test_df['valence_predicted'] = test_df['valence_predicted'].astype(float)
test_df['arousal_predicted'] = test_df['arousal_predicted'].astype(float)

# Prepare the input features and target variables
X_train = train_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']]
y_train_valence = train_df['valence_predicted']
y_train_arousal = train_df['arousal_predicted']
```

```
X_test = test_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']]
y_test_valence = test_df['valence_predicted']
y_test_arousal = test_df['arousal_predicted']

# Create and train the Random Forest models
valence_model = RandomForestRegressor(n_estimators=100, max_depth=10,
random_state=42)
arousal_model = RandomForestRegressor(n_estimators=100, max_depth=10,
random_state=42)

valence_model.fit(X_train, y_train_valence)
arousal_model.fit(X_train, y_train_arousal)

# Make predictions on the test set
valence_pred = valence_model.predict(X_test)
arousal_pred = arousal_model.predict(X_test)

# Save the predictions to a CSV file
predictions_df = pd.DataFrame({
    'valence_pred': valence_pred,
    'arousal_pred': arousal_pred,
    'valence_actual': y_test_valence,
    'arousal_actual': y_test_arousal
})
predictions_df.to_csv('random_forest_predictions.csv', index=False)

# Calculate evaluation metrics
valence_mae = mean_absolute_error(y_test_valence, valence_pred)
valence_mse = mean_squared_error(y_test_valence, valence_pred)
valence_r2 = r2_score(y_test_valence, valence_pred)

arousal_mae = mean_absolute_error(y_test_arousal, arousal_pred)
arousal_mse = mean_squared_error(y_test_arousal, arousal_pred)
arousal_r2 = r2_score(y_test_arousal, arousal_pred)

print("Evaluation Results:")
print(f"Valence - MAE: {valence_mae:.4f}, MSE: {valence_mse:.4f}, R^2:
{valence_r2:.4f}")
print(f"Arousal - MAE: {arousal_mae:.4f}, MSE: {arousal_mse:.4f}, R^2:
{arousal_r2:.4f}")~
```

Gradient Boosting

Script "Gradient Boosting SONG DATASET.py" escrito na linguagem Python, que utiliza as bibliotecas pandas, json, pandas, numpy, sklearn e matplotlib.

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Create a mapping of chord labels to integer values
chord_mapping = {}

def extract_chords_metadata(chunk):
    def extract_num_chords(x):
        if isinstance(x, dict) and 'chordSequence' in x:
            return len(x['chordSequence'])
        else:
            return 0

    def extract_avg_chord_duration(x):
        if isinstance(x, dict) and 'chordSequence' in x and 'duration' in
x and len(x['chordSequence']) > 0:
            return x['duration'] / len(x['chordSequence'])
        else:
            return 0

    def extract_chord_labels(x):
        global chord_mapping
        if isinstance(x, dict) and 'chordSequence' in x:
            labels = [chord['label'] for chord in x['chordSequence'] if
'label' in chord]
            for label in labels:
                if label not in chord_mapping:
                    chord_mapping[label] = len(chord_mapping)
            return [chord_mapping[label] for label in labels]
        else:
            return []

    chunk['num_chords'] =
chunk['chords_metadata'].apply(extract_num_chords)
    chunk['avg_chord_duration'] =
chunk['chords_metadata'].apply(extract_avg_chord_duration)
```

```
    chunk['chord_labels'] =
chunk['chords_metadata'].apply(extract_chord_labels)
    return chunk

# Load the song_extracted_all_entries.json file in batches
batch_size = 1000
train_data = []
test_data = []
total_entries = 0
training_entries = 0

with pd.read_json('song_extracted_all_entries.json', lines=True,
chunksize=batch_size) as reader:
    for i, chunk in enumerate(reader):
        # Check if the required columns are present in the DataFrame
        required_columns = ['_id', 'bpm', 'gain', 'length',
'arousal_predicted', 'valence_predicted', 'chords_metadata']
        missing_columns = [col for col in required_columns if col not in
chunk.columns]

        if missing_columns:
            raise ValueError(f"The following required columns are missing
in the DataFrame: {missing_columns}")

        # Extract chords metadata
        chunk = extract_chords_metadata(chunk)

        # Replace empty string values with np.nan for relevant columns
        chunk['bpm'] = pd.to_numeric(chunk['bpm'], errors='coerce')
        chunk['gain'] = pd.to_numeric(chunk['gain'], errors='coerce')
        chunk['length'] = pd.to_numeric(chunk['length'], errors='coerce')
        chunk['arousal_predicted'] =
chunk['arousal_predicted'].apply(lambda x: np.nan if x == '' else x)
        chunk['valence_predicted'] =
chunk['valence_predicted'].apply(lambda x: np.nan if x == '' else x)

        # Select only the relevant columns
        chunk = chunk[['_id', 'bpm', 'gain', 'length',
'arousal_predicted', 'valence_predicted', 'num_chords',
'avg_chord_duration', 'chord_labels']]

        # Filter rows with missing or zero values
        chunk = chunk[(chunk['bpm'].notnull()) &
(chunk['gain'].notnull()) & (chunk['length'].notnull()) &
(chunk['arousal_predicted'].notnull()) &
(chunk['valence_predicted'].notnull()) & (chunk['num_chords'] != 0) &
(chunk['avg_chord_duration'] != 0)]

        if len(chunk) >= 5:  # Adjust the threshold as needed
```

```
            # Split the current batch into train and test sets
            train_chunk, test_chunk = train_test_split(chunk,
test_size=0.2, random_state=42)

            # Append the train and test data to the respective lists
            train_data.append(train_chunk)
            test_data.append(test_chunk)

            training_entries += len(train_chunk)
        else:
            # If the number of samples is below the threshold, append the
entire chunk to the train data
            train_data.append(chunk)
            training_entries += len(chunk)

        total_entries += len(chunk)
        print(f"Loaded {total_entries} entries")

        # Print the last 5 rows of the chunk
        print(chunk.tail())

        # Print the column names
        print(chunk.columns)

print(f"Data loading completed. Total entries: {total_entries}, Training
entries: {training_entries}")

# Concatenate the train and test data
train_df = pd.concat(train_data, ignore_index=True)
test_df = pd.concat(test_data, ignore_index=True)

print("Train data shape:", train_df.shape)
print("Test data shape:", test_df.shape)

# Convert target variables to float
train_df['valence_predicted'] =
train_df['valence_predicted'].astype(float)
train_df['arousal_predicted'] =
train_df['arousal_predicted'].astype(float)
test_df['valence_predicted'] = test_df['valence_predicted'].astype(float)
test_df['arousal_predicted'] = test_df['arousal_predicted'].astype(float)

# Prepare the input features and target variables
X_train = train_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']]
y_train_valence = train_df['valence_predicted']
y_train_arousal = train_df['arousal_predicted']
```

```
X_test = test_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']]
y_test_valence = test_df['valence_predicted']
y_test_arousal = test_df['arousal_predicted']

# Create and train the Gradient Boosting models
valence_model = GradientBoostingRegressor(n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)
arousal_model = GradientBoostingRegressor(n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)

valence_model.fit(X_train, y_train_valence)
arousal_model.fit(X_train, y_train_arousal)

# Make predictions on the test set
valence_pred = valence_model.predict(X_test)
arousal_pred = arousal_model.predict(X_test)

# Calculate evaluation metrics
valence_mae = mean_absolute_error(y_test_valence, valence_pred)
valence_mse = mean_squared_error(y_test_valence, valence_pred)
valence_r2 = r2_score(y_test_valence, valence_pred)

arousal_mae = mean_absolute_error(y_test_arousal, arousal_pred)
arousal_mse = mean_squared_error(y_test_arousal, arousal_pred)
arousal_r2 = r2_score(y_test_arousal, arousal_pred)

print("Evaluation Results:")
print(f"Valence - MAE: {valence_mae:.4f}, MSE: {valence_mse:.4f}, R^2:
{valence_r2:.4f}")
print(f"Arousal - MAE: {arousal_mae:.4f}, MSE: {arousal_mse:.4f}, R^2:
{arousal_r2:.4f}")
```

SVR

Script "SVR SONG DATASET.py" escrito na linguagem Python, que utiliza as bibliotecas pandas, json, pandas, numpy, sklearn e matplotlib.

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import torch
import torch.nn as nn
```

```
# Create a mapping of chord labels to integer values
chord_mapping = {}

def extract_chords_metadata(chunk):
    def extract_num_chords(x):
        if isinstance(x, dict) and 'chordSequence' in x:
            return len(x['chordSequence'])
        else:
            return 0

    def extract_avg_chord_duration(x):
        if isinstance(x, dict) and 'chordSequence' in x and 'duration' in
x and len(x['chordSequence']) > 0:
            return x['duration'] / len(x['chordSequence'])
        else:
            return 0

    def extract_chord_labels(x):
        global chord_mapping
        if isinstance(x, dict) and 'chordSequence' in x:
            labels = [chord['label'] for chord in x['chordSequence'] if
'label' in chord]
            for label in labels:
                if label not in chord_mapping:
                    chord_mapping[label] = len(chord_mapping)
            return [chord_mapping[label] for label in labels]
        else:
            return []

    chunk['num_chords'] =
chunk['chords_metadata'].apply(extract_num_chords)
    chunk['avg_chord_duration'] =
chunk['chords_metadata'].apply(extract_avg_chord_duration)
    chunk['chord_labels'] =
chunk['chords_metadata'].apply(extract_chord_labels)
    return chunk

# Load the song_extracted_all_entries.json file in batches
batch_size = 1000
train_data = []
test_data = []
total_entries = 0
training_entries = 0

with pd.read_json('song_extracted_all_entries.json', lines=True,
chunksize=batch_size) as reader:
    for i, chunk in enumerate(reader):
        # Check if the required columns are present in the DataFrame
```

```
        required_columns = ['_id', 'bpm', 'gain', 'length',
'arousal_predicted', 'valence_predicted', 'chords_metadata']
        missing_columns = [col for col in required_columns if col not in
chunk.columns]

        if missing_columns:
            raise ValueError(f"The following required columns are missing
in the DataFrame: {missing_columns}")

        # Extract chords metadata
        chunk = extract_chords_metadata(chunk)

        # Select only the relevant columns
        chunk = chunk[['_id', 'bpm', 'gain', 'length',
'arousal_predicted', 'valence_predicted', 'num_chords',
'avg_chord_duration', 'chord_labels']]

        # Convert empty strings to NaN
        chunk = chunk.applymap(lambda x: np.nan if x == '' else x)

        # Drop rows with missing values in relevant columns
        chunk = chunk.dropna(subset=['bpm', 'gain', 'length',
'arousal_predicted', 'valence_predicted', 'num_chords',
'avg_chord_duration'])

        if len(chunk) >= 5:  # Adjust the threshold as needed
            # Split the current batch into train and test sets
            train_chunk, test_chunk = train_test_split(chunk,
test_size=0.2, random_state=42)

            # Append the train and test data to the respective lists
            train_data.append(train_chunk)
            test_data.append(test_chunk)

            training_entries += len(train_chunk)
        else:
            # If the number of samples is below the threshold, append the
entire chunk to the train data
            train_data.append(chunk)
            training_entries += len(chunk)

        total_entries += len(chunk)
        print(f"Loaded {total_entries} entries")

        # Print the last 5 rows of the chunk
        print(chunk.tail())

        # Print the column names
        print(chunk.columns)
```

```
print(f"Data loading completed. Total entries: {total_entries}, Training
entries: {training_entries}")

# Concatenate the train and test data
train_df = pd.concat(train_data, ignore_index=True)
test_df = pd.concat(test_data, ignore_index=True)

print("Data split completed")

# Print the data types of each column
print("Train data types:")
print(train_df.dtypes)
print()
print("Test data types:")
print(test_df.dtypes)
print()

# Check for missing values in relevant columns
print("Train data missing values:")
print(train_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']].isnull().sum())
print()
print("Test data missing values:")
print(test_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']].isnull().sum())
print()

# Set the maximum sequence length for chord labels
max_seq_length = 100

print("Extracting features and labels from the training data...")
train_features = train_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']].values
train_features = train_features.astype(float)  # Convert train_features
to float type
train_valence = train_df['valence_predicted'].values
train_arousal = train_df['arousal_predicted'].values
print("Extraction of training features and labels completed.")

print("Extracting features and labels from the test data...")
test_features = test_df[['bpm', 'gain', 'length', 'num_chords',
'avg_chord_duration']].values
test_features = test_features.astype(float)  # Convert test_features to
float type
test_valence = test_df['valence_predicted'].values
test_arousal = test_df['arousal_predicted'].values
print("Extraction of test features and labels completed.")
```

```
# Set the device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Convert data to PyTorch tensors and move to the GPU
train_features_tensor =
torch.from_numpy(train_features).float().to(device)
train_valence_tensor = torch.from_numpy(train_valence).float().to(device)
train_arousal_tensor = torch.from_numpy(train_arousal).float().to(device)
test_features_tensor = torch.from_numpy(test_features).float().to(device)
test_valence_tensor = torch.from_numpy(test_valence).float().to(device)
test_arousal_tensor = torch.from_numpy(test_arousal).float().to(device)

# Define the SVR models using PyTorch with RBF kernel
class SVRModel(nn.Module):
    def __init__(self, input_dim, gamma):
        super(SVRModel, self).__init__()
        self.gamma = gamma
        self.linear = nn.Linear(input_dim, 1)

    def kernel(self, x1, x2):
        # Radial Basis Function (RBF) kernel
        dist = torch.cdist(x1.unsqueeze(1), x2.unsqueeze(0), p=2)
        return torch.exp(-self.gamma * dist ** 2)

    def forward(self, x):
        K = self.kernel(x, self.linear.weight)
        return K @ self.linear.weight + self.linear.bias

# Define the SVR models with RBF kernel
input_dim = train_features.shape[1]
gamma = 0.1  # Adjust the gamma value as needed
valence_model = SVRModel(input_dim, gamma).to(device)
arousal_model = SVRModel(input_dim, gamma).to(device)

# Define the loss function and optimizer
criterion = nn.MSELoss()
valence_optimizer = torch.optim.Adam(valence_model.parameters())
arousal_optimizer = torch.optim.Adam(arousal_model.parameters())

# Train the SVR models
num_epochs = 100
batch_size = 1024

print("Training the valence model...")
for epoch in range(num_epochs):
    for i in range(0, len(train_features_tensor), batch_size):
        batch_features = train_features_tensor[i:i+batch_size]
        batch_valence = train_valence_tensor[i:i+batch_size]
```

```
        valence_optimizer.zero_grad()
        valence_pred = valence_model(batch_features)
        loss = criterion(valence_pred, batch_valence.view(-1, 1))
        loss.backward()
        valence_optimizer.step()
print("Valence model training completed.")

print("Training the arousal model...")
for epoch in range(num_epochs):
    for i in range(0, len(train_features_tensor), batch_size):
        batch_features = train_features_tensor[i:i+batch_size]
        batch_arousal = train_arousal_tensor[i:i+batch_size]

        arousal_optimizer.zero_grad()
        arousal_pred = arousal_model(batch_features)
        loss = criterion(arousal_pred, batch_arousal.view(-1, 1))
        loss.backward()
        arousal_optimizer.step()
print("Arousal model training completed.")

# Predict valence and arousal values for the test data
print("Predicting valence values for the test data...")
with torch.no_grad():
    valence_pred =
valence_model(test_features_tensor).cpu().numpy().flatten()[:len(test_val
ence)]
print("Valence prediction completed.")

print("Predicting arousal values for the test data...")
with torch.no_grad():
    arousal_pred =
arousal_model(test_features_tensor).cpu().numpy().flatten()[:len(test_aro
usal)]
print("Arousal prediction completed.")

# Calculate evaluation metrics
print("Calculating evaluation metrics...")
valence_mae = mean_absolute_error(test_valence, valence_pred)
valence_mse = mean_squared_error(test_valence, valence_pred)
valence_r2 = r2_score(test_valence, valence_pred)

arousal_mae = mean_absolute_error(test_arousal, arousal_pred)
arousal_mse = mean_squared_error(test_arousal, arousal_pred)
arousal_r2 = r2_score(test_arousal, arousal_pred)
print("Evaluation metrics calculation completed.")

print("Evaluation Results:")
```

```
print(f"Valence - MAE: {valence_mae:.4f}, MSE: {valence_mse:.4f}, R^2:
{valence_r2:.4f}")
print(f"Arousal - MAE: {arousal_mae:.4f}, MSE: {arousal_mse:.4f}, R^2:
{arousal_r2:.4f}")
```

A.3 Modelos de *Deep Learning*

CNN + Mecanismo de Atenção

Script "CNN + Attention Model SONG DATASET.py" escrito na linguagem Python, que utiliza as cm json, pandas, sklearn, torch, transformers, iertools e keras.

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import torch
from transformers import AdamW
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
from torch.optim import Adam
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from transformers import BertTokenizer, BertModel

# Initialize the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Load the song_extracted_all_entries.json file in batches
batch_size = 1000
train_data = []
test_data = []

with pd.read_json('song_extracted_all_entries.json', lines=True,
chunksize=batch_size) as reader:
    total_entries = 0
    for i, chunk in enumerate(reader):
        # Check if the required columns are present in the DataFrame
        required_columns = ['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']
        missing_columns = [col for col in required_columns if col not in
chunk.columns]

        if missing_columns:
```

```
            raise ValueError(f"The following required columns are missing
in the DataFrame: {missing_columns}")

        # Fill missing values with appropriate placeholders or default
values
        chunk['lyrics'] = chunk['lyrics'].fillna('')
        chunk['arousal_predicted'] = chunk['arousal_predicted'].fillna(0)
        chunk['valence_predicted'] = chunk['valence_predicted'].fillna(0)

        # Convert 'arousal_predicted' and 'valence_predicted' to float
        chunk['arousal_predicted'] =
chunk['arousal_predicted'].astype(float)
        chunk['valence_predicted'] =
chunk['valence_predicted'].astype(float)

        # Select only the relevant columns
        chunk = chunk[['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']]

        # Filter out entries without lyrics
        chunk = chunk[chunk['lyrics'] != '']

        # Tokenize the lyrics using the BERT tokenizer
        def tokenize_lyrics(lyrics):
            return tokenizer.encode(lyrics, add_special_tokens=True,
max_length=512, truncation=True, padding='max_length')

        chunk['input_ids'] = chunk['lyrics'].apply(tokenize_lyrics)

        # Split the current batch into train and test sets
        train_chunk, test_chunk = train_test_split(chunk, test_size=0.2,
random_state=42)

        # Append the train and test data to the respective lists
        train_data.append(train_chunk)
        test_data.append(test_chunk)

        total_entries += len(chunk)
        print(f"Loaded {total_entries} entries")

        # Print the last 5 rows of the chunk
        print(chunk.tail())

        # Print the column names
        print(chunk.columns)

        # Check if the desired number of entries has been processed
        if total_entries >= 10000:
            break
```

```
print("Data loading completed.")

# Combine the train and test data from all batches
train_df = pd.concat(train_data, ignore_index=True)
test_df = pd.concat(test_data, ignore_index=True)

print("Lyrics tokenization and data split completed")

# Filter out samples with missing valence or arousal values
train_df = train_df.dropna(subset=['valence_predicted',
'arousal_predicted'])
test_df = test_df.dropna(subset=['valence_predicted',
'arousal_predicted'])

# Convert 'valence_predicted' and 'arousal_predicted' columns to numeric
train_df['valence_predicted'] =
pd.to_numeric(train_df['valence_predicted'], errors='coerce')
train_df['arousal_predicted'] =
pd.to_numeric(train_df['arousal_predicted'], errors='coerce')
test_df['valence_predicted'] =
pd.to_numeric(test_df['valence_predicted'], errors='coerce')
test_df['arousal_predicted'] =
pd.to_numeric(test_df['arousal_predicted'], errors='coerce')

# Create a PyTorch Dataset
class EmotionDataset(Dataset):
    def __init__(self, dataframe):
        self.dataframe = dataframe

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        row = self.dataframe.iloc[index]
        input_ids = torch.tensor(row['input_ids'], dtype=torch.long)
        attention_mask = torch.ones_like(input_ids)
        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'valence': torch.tensor(row['valence_predicted'],
dtype=torch.float),
            'arousal': torch.tensor(row['arousal_predicted'],
dtype=torch.float)
        }

# Define the model architecture
class EmotionRegressionModel(nn.Module):
    def __init__(self, hidden_dim, output_dim):
```

```
        super(EmotionRegressionModel, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased',
gradient_checkpointing=True)
        self.fc1 = nn.Linear(self.bert.config.hidden_size, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, input_ids, attention_mask):
        print("Forward pass called")  # Add this line
        bert_outputs = self.bert(input_ids,
attention_mask=attention_mask)
        last_hidden = bert_outputs.last_hidden_state[:, 0, :]
        hidden = self.fc1(last_hidden)
        output = self.fc2(hidden)
        return output

# Set device
device = torch.device('cuda')
print(f"Device: {device}")

# Instantiate the model
hidden_dim = 256
output_dim = 2
model = EmotionRegressionModel(hidden_dim, output_dim)
model.to(device)
print("Model instantiation completed")

# Define loss function and optimizer
criterion = nn.MSELoss().to(device)
optimizer = AdamW(model.parameters(), lr=1e-5)

# Training loop
num_epochs = 10
batch_size = 32
accumulation_steps = 4

train_dataset = EmotionDataset(train_df)
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

for epoch in range(num_epochs):
    print(f"Epoch {epoch + 1}/{num_epochs}")
    model.train()

    for batch_idx, batch in enumerate(train_loader):
        print(f"Processing batch {batch_idx+1}")  # Add this line
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)
```

```
        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs, torch.stack([valence, arousal], dim=1))
        loss = loss / accumulation_steps
        loss.backward()

        if (batch_idx + 1) % accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()

        print(f"Epoch [{epoch+1}/{num_epochs}], Batch
[{batch_idx+1}/{len(train_loader)}], Loss: {loss.item():.4f}")

# Evaluation
model.eval()
valence_true = []
arousal_true = []
valence_pred = []
arousal_pred = []

test_dataset = EmotionDataset(test_df)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)

        outputs = model(input_ids, attention_mask)
        valence_pred.extend(outputs[:, 0].cpu().numpy())
        arousal_pred.extend(outputs[:, 1].cpu().numpy())
        valence_true.extend(valence.cpu().numpy())
        arousal_true.extend(arousal.cpu().numpy())

valence_true = np.array(valence_true)
arousal_true = np.array(arousal_true)
valence_pred = np.array(valence_pred)
arousal_pred = np.array(arousal_pred)

# Save predictions to a CSV file
predictions_df = pd.DataFrame({'valence_true': valence_true,
                               'arousal_true': arousal_true,
                               'valence_pred': valence_pred,
                               'arousal_pred': arousal_pred})
predictions_df.to_csv('predictions.csv', index=False)

valence_mae = mean_absolute_error(valence_true, valence_pred)
```

```
valence_mse = mean_squared_error(valence_true, valence_pred)
valence_r2 = r2_score(valence_true, valence_pred)

arousal_mae = mean_absolute_error(arousal_true, arousal_pred)
arousal_mse = mean_squared_error(arousal_true, arousal_pred)
arousal_r2 = r2_score(arousal_true, arousal_pred)

print("Valence Evaluation:")
print("MAE:", valence_mae)
print("MSE:", valence_mse)
print("R2 Score:", valence_r2)

print("\nArousal Evaluation:")
print("MAE:", arousal_mae)
print("MSE:", arousal_mse)
print("R2 Score:", arousal_r2)
```

LSTM

Script "LSTM SONG DATASET.py" escrito na linguagem Python, que utiliza as bibliotecas json, pandas, numpy, sklearn, torch, transformers e sklearn.

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import torch
from transformers import AdamW
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
from torch.optim import Adam
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from transformers import BertTokenizer

# Initialize the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Load the song_extracted_all_entries.json file in batches
batch_size = 1000
train_data = []
test_data = []
```

```
with pd.read_json('song_extracted_all_entries.json', lines=True,
chunksize=batch_size) as reader:
    total_entries = 0
    for i, chunk in enumerate(reader):
        # Check if the required columns are present in the DataFrame
        required_columns = ['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']
        missing_columns = [col for col in required_columns if col not in
chunk.columns]

        if missing_columns:
            raise ValueError(f"The following required columns are missing
in the DataFrame: {missing_columns}")

        # Fill missing values with appropriate placeholders or default
values
        chunk['lyrics'] = chunk['lyrics'].fillna('')
        chunk['arousal_predicted'] = chunk['arousal_predicted'].fillna(0)
        chunk['valence_predicted'] = chunk['valence_predicted'].fillna(0)

        # Convert 'arousal_predicted' and 'valence_predicted' to float
        chunk['arousal_predicted'] =
chunk['arousal_predicted'].astype(float)
        chunk['valence_predicted'] =
chunk['valence_predicted'].astype(float)

        # Select only the relevant columns
        chunk = chunk[['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']]

        # Filter out entries without lyrics
        chunk = chunk[chunk['lyrics'] != '']

        # Tokenize the lyrics using the BERT tokenizer
        def tokenize_lyrics(lyrics):
            return tokenizer.encode(lyrics, add_special_tokens=True,
max_length=512, truncation=True, padding='max_length')

        chunk['input_ids'] = chunk['lyrics'].apply(tokenize_lyrics)

        # Split the current batch into train and test sets
        train_chunk, test_chunk = train_test_split(chunk, test_size=0.2,
random_state=42)

        # Append the train and test data to the respective lists
        train_data.append(train_chunk)
        test_data.append(test_chunk)

        total_entries += len(chunk)
```

```
        print(f"Loaded {total_entries} entries")

        # Print the last 5 rows of the chunk
        print(chunk.tail())

        # Print the column names
        print(chunk.columns)

        # Check if the desired number of entries has been processed
        if total_entries >= 10000:
            break

print("Data loading completed.")

# Combine the train and test data from all batches
train_df = pd.concat(train_data, ignore_index=True)
test_df = pd.concat(test_data, ignore_index=True)

print("Lyrics tokenization and data split completed")

# Filter out samples with missing valence or arousal values
train_df = train_df.dropna(subset=['valence_predicted',
'arousal_predicted'])
test_df = test_df.dropna(subset=['valence_predicted',
'arousal_predicted'])

# Convert 'valence_predicted' and 'arousal_predicted' columns to numeric
train_df['valence_predicted'] =
pd.to_numeric(train_df['valence_predicted'], errors='coerce')
train_df['arousal_predicted'] =
pd.to_numeric(train_df['arousal_predicted'], errors='coerce')
test_df['valence_predicted'] =
pd.to_numeric(test_df['valence_predicted'], errors='coerce')
test_df['arousal_predicted'] =
pd.to_numeric(test_df['arousal_predicted'], errors='coerce')

# Create a PyTorch Dataset
class EmotionDataset(Dataset):
    def __init__(self, dataframe):
        self.dataframe = dataframe

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        row = self.dataframe.iloc[index]
        input_ids = torch.tensor(row['input_ids'], dtype=torch.long)
        attention_mask = torch.ones_like(input_ids)
        return {
```

```
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'valence': torch.tensor(row['valence_predicted'],
dtype=torch.float),
            'arousal': torch.tensor(row['arousal_predicted'],
dtype=torch.float)
        }

# Define the model architecture
class EmotionRegressionModel(nn.Module):
    def __init__(self, hidden_dim, output_dim):
        super(EmotionRegressionModel, self).__init__()
        vocab_size = len(tokenizer.get_vocab())
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.lstm = nn.LSTM(input_size=hidden_dim,
hidden_size=hidden_dim, num_layers=2, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, input_ids, attention_mask):
        print("Forward pass called")  # Add this line
        embedded = self.embedding(input_ids)
        _, (hidden, _) = self.lstm(embedded)
        hidden = hidden[-1]  # Take the last hidden state
        output = self.fc(hidden)
        return output

# Set device
device = torch.device('cuda')
print(f"Device: {device}")

# Instantiate the model
hidden_dim = 256
output_dim = 2
model = EmotionRegressionModel(hidden_dim, output_dim)
model.to(device)
print("Model instantiation completed")

# Define loss function and optimizer
criterion = nn.MSELoss().to(device)
optimizer = AdamW(model.parameters(), lr=1e-5)

# Training loop
num_epochs = 10
batch_size = 32
accumulation_steps = 4

train_dataset = EmotionDataset(train_df)
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
```

```
for epoch in range(num_epochs):
    print(f"Epoch {epoch + 1}/{num_epochs}")
    model.train()

    for batch_idx, batch in enumerate(train_loader):
        print(f"Processing batch {batch_idx+1}")  # Add this line
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)

        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs, torch.stack([valence, arousal], dim=1))
        loss = loss / accumulation_steps
        loss.backward()

        if (batch_idx + 1) % accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()

        print(f"Epoch [{epoch+1}/{num_epochs}], Batch
[{batch_idx+1}/{len(train_loader)}], Loss: {loss.item():.4f}")

# Evaluation
model.eval()
valence_true = []
arousal_true = []
valence_pred = []
arousal_pred = []

test_dataset = EmotionDataset(test_df)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)

        outputs = model(input_ids, attention_mask)
        valence_pred.extend(outputs[:, 0].cpu().numpy())
        arousal_pred.extend(outputs[:, 1].cpu().numpy())
        valence_true.extend(valence.cpu().numpy())
        arousal_true.extend(arousal.cpu().numpy())

valence_true = np.array(valence_true)
arousal_true = np.array(arousal_true)
```

```
valence_pred = np.array(valence_pred)
arousal_pred = np.array(arousal_pred)

valence_mae = mean_absolute_error(valence_true, valence_pred)
valence_mse = mean_squared_error(valence_true, valence_pred)
valence_r2 = r2_score(valence_true, valence_pred)

arousal_mae = mean_absolute_error(arousal_true, arousal_pred)
arousal_mse = mean_squared_error(arousal_true, arousal_pred)
arousal_r2 = r2_score(arousal_true, arousal_pred)

print("Valence Evaluation:")
print("MAE:", valence_mae)
print("MSE:", valence_mse)
print("R2 Score:", valence_r2)

print("\nArousal Evaluation:")
print("MAE:", arousal_mae)
print("MSE:", arousal_mse)
print("R2 Score:", arousal_r2)
```

BiLSTM + Mecanismo de Atenção

Script "BiLSTM + Attention Model SONG DATASET.py" escrito na linguagem Python, que utiliza as bibliotecas json, pandas, numpy, sklearn, torch, transformers e sklearn.

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import torch
from transformers import AdamW
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
from torch.optim import Adam
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from transformers import BertTokenizer, BertModel

# Initialize the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Load the song_extracted_all_entries.json file in batches
batch_size = 1000
train_data = []
test_data = []
```

```
with pd.read_json('song_extracted_all_entries.json', lines=True,
chunksize=batch_size) as reader:
    total_entries = 0
    for i, chunk in enumerate(reader):
        # Check if the required columns are present in the DataFrame
        required_columns = ['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']
        missing_columns = [col for col in required_columns if col not in
chunk.columns]

        if missing_columns:
            raise ValueError(f"The following required columns are missing
in the DataFrame: {missing_columns}")

        # Fill missing values with appropriate placeholders or default
values
        chunk['lyrics'] = chunk['lyrics'].fillna('')
        chunk['arousal_predicted'] = chunk['arousal_predicted'].fillna(0)
        chunk['valence_predicted'] = chunk['valence_predicted'].fillna(0)

        # Convert 'arousal_predicted' and 'valence_predicted' to float
        chunk['arousal_predicted'] =
chunk['arousal_predicted'].astype(float)
        chunk['valence_predicted'] =
chunk['valence_predicted'].astype(float)

        # Select only the relevant columns
        chunk = chunk[['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']]

        # Filter out entries without lyrics
        chunk = chunk[chunk['lyrics'] != '']

        # Tokenize the lyrics using the BERT tokenizer
        def tokenize_lyrics(lyrics):
            return tokenizer.encode(lyrics, add_special_tokens=True,
max_length=512, truncation=True, padding='max_length')

        chunk['input_ids'] = chunk['lyrics'].apply(tokenize_lyrics)

        # Split the current batch into train and test sets
        train_chunk, test_chunk = train_test_split(chunk, test_size=0.2,
random_state=42)

        # Append the train and test data to the respective lists
        train_data.append(train_chunk)
        test_data.append(test_chunk)
```

```
        total_entries += len(chunk)
        print(f"Loaded {total_entries} entries")

        # Print the last 5 rows of the chunk
        print(chunk.tail())

        # Print the column names
        print(chunk.columns)

        # Check if the desired number of entries has been processed
        if total_entries >= 10000:
            break

print("Data loading completed.")

# Combine the train and test data from all batches
train_df = pd.concat(train_data, ignore_index=True)
test_df = pd.concat(test_data, ignore_index=True)

print("Lyrics tokenization and data split completed")

# Filter out samples with missing valence or arousal values
train_df = train_df.dropna(subset=['valence_predicted',
'arousal_predicted'])
test_df = test_df.dropna(subset=['valence_predicted',
'arousal_predicted'])

# Convert 'valence_predicted' and 'arousal_predicted' columns to numeric
train_df['valence_predicted'] =
pd.to_numeric(train_df['valence_predicted'], errors='coerce')
train_df['arousal_predicted'] =
pd.to_numeric(train_df['arousal_predicted'], errors='coerce')
test_df['valence_predicted'] =
pd.to_numeric(test_df['valence_predicted'], errors='coerce')
test_df['arousal_predicted'] =
pd.to_numeric(test_df['arousal_predicted'], errors='coerce')

# Create a PyTorch Dataset
class EmotionDataset(Dataset):
    def __init__(self, dataframe):
        self.dataframe = dataframe

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        row = self.dataframe.iloc[index]
        input_ids = torch.tensor(row['input_ids'], dtype=torch.long)
        attention_mask = torch.ones_like(input_ids)
```

```
        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'valence': torch.tensor(row['valence_predicted'],
dtype=torch.float),
            'arousal': torch.tensor(row['arousal_predicted'],
dtype=torch.float)
        }

# Define the model architecture
class EmotionRegressionModel(nn.Module):
    def __init__(self, hidden_dim, output_dim):
        super(EmotionRegressionModel, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased',
gradient_checkpointing=True)
        self.bilstm = nn.LSTM(self.bert.config.hidden_size, hidden_dim,
num_layers=2, bidirectional=True, batch_first=True)
        self.attention = nn.MultiheadAttention(hidden_dim * 2,
num_heads=8)
        self.fc = nn.Linear(hidden_dim * 2, output_dim)

    def forward(self, input_ids, attention_mask):
        print("Forward pass called")  # Add this line
        bert_outputs = self.bert(input_ids,
attention_mask=attention_mask)
        last_hidden = bert_outputs.last_hidden_state
        lstm_outputs, _ = self.bilstm(last_hidden)
        attn_output, _ = self.attention(lstm_outputs, lstm_outputs,
lstm_outputs)
        pooled_output = attn_output.mean(dim=1)
        output = self.fc(pooled_output)
        return output

# Set device
device = torch.device('cuda')
print(f"Device: {device}")

# Instantiate the model
hidden_dim = 256
output_dim = 2
model = EmotionRegressionModel(hidden_dim, output_dim)
model.to(device)
print("Model instantiation completed")

# Define loss function and optimizer
criterion = nn.MSELoss().to(device)
optimizer = AdamW(model.parameters(), lr=1e-5)

# Training loop
```

```
num_epochs = 10
batch_size = 32
accumulation_steps = 4

train_dataset = EmotionDataset(train_df)
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

for epoch in range(num_epochs):
    print(f"Epoch {epoch + 1}/{num_epochs}")
    model.train()

    for batch_idx, batch in enumerate(train_loader):
        print(f"Processing batch {batch_idx+1}")  # Add this line
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)

        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs, torch.stack([valence, arousal], dim=1))
        loss = loss / accumulation_steps
        loss.backward()

        if (batch_idx + 1) % accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()

        print(f"Epoch [{epoch+1}/{num_epochs}], Batch
[{batch_idx+1}/{len(train_loader)}], Loss: {loss.item():.4f}")

# Evaluation
model.eval()
valence_true = []
arousal_true = []
valence_pred = []
arousal_pred = []

test_dataset = EmotionDataset(test_df)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)

        outputs = model(input_ids, attention_mask)
```

```
        valence_pred.extend(outputs[:, 0].cpu().numpy())
        arousal_pred.extend(outputs[:, 1].cpu().numpy())
        valence_true.extend(valence.cpu().numpy())
        arousal_true.extend(arousal.cpu().numpy())

valence_true = np.array(valence_true)
arousal_true = np.array(arousal_true)
valence_pred = np.array(valence_pred)
arousal_pred = np.array(arousal_pred)

valence_mae = mean_absolute_error(valence_true, valence_pred)
valence_mse = mean_squared_error(valence_true, valence_pred)
valence_r2 = r2_score(valence_true, valence_pred)

arousal_mae = mean_absolute_error(arousal_true, arousal_pred)
arousal_mse = mean_squared_error(arousal_true, arousal_pred)
arousal_r2 = r2_score(arousal_true, arousal_pred)

print("Valence Evaluation:")
print("MAE:", valence_mae)
print("MSE:", valence_mse)
print("R2 Score:", valence_r2)

print("\nArousal Evaluation:")
print("MAE:", arousal_mae)
print("MSE:", arousal_mse)
print("R2 Score:", arousal_r2)
```

HAN

Script "HAN SONG DATASET.py" escrito na linguagem Python, que utiliza as bibliotecas json, pandas, numpy, sklearn, torch, transformers e sklearn.

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import torch
from transformers import AdamW
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
from torch.optim import Adam
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from transformers import BertTokenizer, BertModel
```

```
# Initialize the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Load the song_extracted_all_entries.json file in batches
batch_size = 1000
train_data = []
test_data = []

with pd.read_json('song_extracted_all_entries.json', lines=True,
chunksize=batch_size) as reader:
    total_entries = 0
    for i, chunk in enumerate(reader):
        # Check if the required columns are present in the DataFrame
        required_columns = ['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']
        missing_columns = [col for col in required_columns if col not in
chunk.columns]

        if missing_columns:
            raise ValueError(f"The following required columns are missing
in the DataFrame: {missing_columns}")

        # Fill missing values with appropriate placeholders or default
values
        chunk['lyrics'] = chunk['lyrics'].fillna('')
        chunk['arousal_predicted'] = chunk['arousal_predicted'].fillna(0)
        chunk['valence_predicted'] = chunk['valence_predicted'].fillna(0)

        # Convert 'arousal_predicted' and 'valence_predicted' to float
        chunk['arousal_predicted'] =
chunk['arousal_predicted'].astype(float)
        chunk['valence_predicted'] =
chunk['valence_predicted'].astype(float)

        # Select only the relevant columns
        chunk = chunk[['_id', 'lyrics', 'arousal_predicted',
'valence_predicted']]

        # Filter out entries without lyrics
        chunk = chunk[chunk['lyrics'] != '']

        # Tokenize the lyrics using the BERT tokenizer
        def tokenize_lyrics(lyrics):
            return tokenizer.encode(lyrics, add_special_tokens=True,
max_length=512, truncation=True, padding='max_length')

        chunk['input_ids'] = chunk['lyrics'].apply(tokenize_lyrics)

        # Split the current batch into train and test sets
```

```
        train_chunk, test_chunk = train_test_split(chunk, test_size=0.2,
random_state=42)

        # Append the train and test data to the respective lists
        train_data.append(train_chunk)
        test_data.append(test_chunk)

        total_entries += len(chunk)
        print(f"Loaded {total_entries} entries")

        # Print the last 5 rows of the chunk
        print(chunk.tail())

        # Print the column names
        print(chunk.columns)

        # Check if the desired number of entries has been processed
        if total_entries >= 10000:
            break

print("Data loading completed.")

# Combine the train and test data from all batches
train_df = pd.concat(train_data, ignore_index=True)
test_df = pd.concat(test_data, ignore_index=True)

print("Lyrics tokenization and data split completed")

# Filter out samples with missing valence or arousal values
train_df = train_df.dropna(subset=['valence_predicted',
'arousal_predicted'])
test_df = test_df.dropna(subset=['valence_predicted',
'arousal_predicted'])

# Convert 'valence_predicted' and 'arousal_predicted' columns to numeric
train_df['valence_predicted'] =
pd.to_numeric(train_df['valence_predicted'], errors='coerce')
train_df['arousal_predicted'] =
pd.to_numeric(train_df['arousal_predicted'], errors='coerce')
test_df['valence_predicted'] =
pd.to_numeric(test_df['valence_predicted'], errors='coerce')
test_df['arousal_predicted'] =
pd.to_numeric(test_df['arousal_predicted'], errors='coerce')

# Create a PyTorch Dataset
class EmotionDataset(Dataset):
    def __init__(self, dataframe):
        self.dataframe = dataframe
```

```
    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        row = self.dataframe.iloc[index]
        input_ids = torch.tensor(row['input_ids'], dtype=torch.long)
        attention_mask = torch.ones_like(input_ids)
        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'valence': torch.tensor(row['valence_predicted'],
dtype=torch.float),
            'arousal': torch.tensor(row['arousal_predicted'],
dtype=torch.float)
        }

# Define the HAN model architecture
class HANModel(nn.Module):
    def __init__(self, hidden_dim, output_dim):
        super(HANModel, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased',
gradient_checkpointing=True)
        self.attention = nn.Linear(self.bert.config.hidden_size, 1)
        self.gru = nn.GRU(self.bert.config.hidden_size, hidden_dim,
batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, input_ids, attention_mask):
        print("Forward pass called")  # Add this line
        bert_outputs = self.bert(input_ids,
attention_mask=attention_mask)
        hidden_states = bert_outputs.last_hidden_state

        # Word-level attention
        word_attention_weights =
torch.softmax(self.attention(hidden_states), dim=1)
        word_level_representation = torch.sum(hidden_states *
word_attention_weights, dim=1)

        # Sentence-level encoding
        _, sentence_representation =
self.gru(word_level_representation.unsqueeze(0))
        sentence_representation = sentence_representation.squeeze(0)

        # Final prediction
        output = self.fc(sentence_representation)
        return output

# Set device
```

```
device = torch.device('cuda')
print(f"Device: {device}")

# Instantiate the HAN model
hidden_dim = 256
output_dim = 2
model = HANModel(hidden_dim, output_dim)
model.to(device)
print("Model instantiation completed")

# Define loss function and optimizer
criterion = nn.MSELoss().to(device)
optimizer = AdamW(model.parameters(), lr=1e-5)

# Training loop
num_epochs = 10
batch_size = 32
accumulation_steps = 4

train_dataset = EmotionDataset(train_df)
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

for epoch in range(num_epochs):
    print(f"Epoch {epoch + 1}/{num_epochs}")
    model.train()

    for batch_idx, batch in enumerate(train_loader):
        print(f"Processing batch {batch_idx+1}")  # Add this line
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)

        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs, torch.stack([valence, arousal], dim=1))
        loss = loss / accumulation_steps
        loss.backward()

        if (batch_idx + 1) % accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()

        print(f"Epoch [{epoch+1}/{num_epochs}], Batch
[{batch_idx+1}/{len(train_loader)}], Loss: {loss.item():.4f}")

# Evaluation
model.eval()
valence_true = []
```

```
arousal_true = []
valence_pred = []
arousal_pred = []

test_dataset = EmotionDataset(test_df)
test_loader = DataLoader(test_dataset, batch_size=1)  # Set batch_size to
1

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valence = batch['valence'].to(device)
        arousal = batch['arousal'].to(device)

        outputs = model(input_ids, attention_mask)
        valence_pred.append(outputs[0, 0].item())
        arousal_pred.append(outputs[0, 1].item())
        valence_true.append(valence.item())
        arousal_true.append(arousal.item())

valence_mae = mean_absolute_error(valence_true, valence_pred)
valence_mse = mean_squared_error(valence_true, valence_pred)
valence_r2 = r2_score(valence_true, valence_pred)

arousal_mae = mean_absolute_error(arousal_true, arousal_pred)
arousal_mse = mean_squared_error(arousal_true, arousal_pred)
arousal_r2 = r2_score(arousal_true, arousal_pred)

print("Valence Evaluation:")
print("MAE:", valence_mae)
print("MSE:", valence_mse)
print("R2 Score:", valence_r2)

print("\nArousal Evaluation:")
print("MAE:", arousal_mae)
print("MSE:", arousal_mse)
print("R2 Score:", arousal_r2)
```

A.4 Modelo de Fusão

Script "Fusion Model.py" escrito na linguagem Python, que utiliza as bibliotecas json, pandas e sklearn.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Load the random forest predictions
rf_predictions = pd.read_csv('random_forest_predictions.csv')

# Load the CNN predictions
cnn_predictions = pd.read_csv('cnn_predictions.csv')

# Ensure both files have the same number of entries
rf_predictions = rf_predictions.sample(n=len(cnn_predictions),
random_state=42)

# Merge the predictions based on the actual values
merged_predictions = pd.merge(rf_predictions, cnn_predictions,
left_on=['valence_actual', 'arousal_actual'],
                              right_on=['valence_true', 'arousal_true'],
how='inner')

# Prepare the input features and target variables
X = merged_predictions[['valence_pred_x', 'arousal_pred_x',
'valence_pred_y', 'arousal_pred_y']]
y_valence = merged_predictions['valence_actual']
y_arousal = merged_predictions['arousal_actual']

# Split the data into training and testing sets
X_train, X_test, y_train_valence, y_test_valence, y_train_arousal,
y_test_arousal = train_test_split(
    X, y_valence, y_arousal, test_size=0.2, random_state=42)

# Create the base models
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
lr_model = LinearRegression()

# Train the base models
rf_model.fit(X_train[['valence_pred_x', 'arousal_pred_x']],
y_train_valence)
```

```
lr_model.fit(X_train[['valence_pred_y', 'arousal_pred_y']],
y_train_arousal)

# Make predictions using the base models
rf_pred_valence = rf_model.predict(X_test[['valence_pred_x',
'arousal_pred_x']])
lr_pred_arousal = lr_model.predict(X_test[['valence_pred_y',
'arousal_pred_y']])

# Combine the predictions from the base models
ensemble_predictions = pd.DataFrame({'rf_pred_valence': rf_pred_valence,
'lr_pred_arousal': lr_pred_arousal})

# Create the meta-model
meta_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the meta-model
meta_model.fit(ensemble_predictions, pd.concat([y_test_valence,
y_test_arousal], axis=1))

# Make predictions using the meta-model
final_predictions = meta_model.predict(ensemble_predictions)

# Evaluate the performance of the fusion model
valence_mae = mean_absolute_error(y_test_valence, final_predictions[:,
0])
valence_mse = mean_squared_error(y_test_valence, final_predictions[:, 0])
valence_r2 = r2_score(y_test_valence, final_predictions[:, 0])

arousal_mae = mean_absolute_error(y_test_arousal, final_predictions[:,
1])
arousal_mse = mean_squared_error(y_test_arousal, final_predictions[:, 1])
arousal_r2 = r2_score(y_test_arousal, final_predictions[:, 1])

print("Fusion Model Evaluation:")
print(f"Valence - MAE: {valence_mae:.4f}, MSE: {valence_mse:.4f}, R^2:
{valence_r2:.4f}")
print(f"Arousal - MAE: {arousal_mae:.4f}, MSE: {arousal_mse:.4f}, R^2:
{arousal_r2:.4f}")
```