Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Tiago Marques António

MER: Estudo e restruturação de um sistema de reconhecimento emocional em música áudio usando o YouTube

Relatório de Projeto de Mestrado

Orientado por:

Professor Renato Panda, Instituto Politécnico de Tomar

Relatório apresentado ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre em
Engenharia Informática – Internet das Coisas

“O céu é o limite.”

Tiago António

RESUMO

O Reconhecimento de Emoções em Música (MER) é uma área de investigação recente, derivada da área de Recuperação de Informações em Música (MIR), que tem vindo a ganhar importância ao longo dos anos. A música está presente na história do ser humano desde que há memória, sendo utilizada nas mais diversas situações, desde entretenimento até fins mais sérios, como na medicina. Não é novidade que a música está intimamente ligada às emoções, sendo utilizada para as transmitir aos ouvintes, podendo manifestar-se de diferentes modos a nível físico e psicológico. Embora os mecanismos que relacionam a música e a emoção sejam ainda hoje bastante incompreendidos, existe vontade por parte dos investigadores da área em melhor compreender estas relações e do ponto de vista mais prático, criar sistemas que consigam identificar as emoções presentes nas músicas.

O processo de identificação de emoções em sinais musicais utilizando abordagens automatizadas é deveras complexo, demorado e delicado, ligando várias áreas de conhecimento, como a psicologia, onde estão abrangidas as emoções, a área de computação, onde é realizado todo o processamento de sinal e a classificação de emoções, ou ainda a necessidade de ter alguns conhecimentos gerais sobre teoria musical. A quantidade de plataformas que atualmente demonstra este conceito é mínima, sendo na sua maioria provas de conceito com fins académicos.

Neste trabalho foi desenvolvido um sistema robusto e escalável de MER, capaz de fazer o reconhecimento de emoções transmitidas em música, adaptando o modelo emocional de *Russell* para quatro classes: alegre, tensa, triste e calma. Este sistema partiu das lições retiradas de uma prova-de-conceito criada num projeto anterior, cuja finalidade foi perceber a exequibilidade de combinar uma abordagem MER com conceitos avançados de desenvolvimento de software. Desta, foi apenas aproveitada uma parte da aplicação *web*, sendo refeita toda a lógica de reconhecimento emocional, aumentando significativamente a complexidade e capacidade do sistema.

O sistema atual integra vários conceitos do estado de arte na área de MER para áudio e letras de músicas (*lyrics*), utilizando a plataforma YouTube com fonte principal de dados. Foi

treinado um classificador para a componente da letra, que permite prever uma única emoção para a letra musical através de caraterísticas extraídas da mesma. Foram desenvolvidos quatro classificadores distintos para a componente de áudio, que permitem obter múltiplas emoções para uma mesma música áudio completa, devolvendo ainda assim uma única classe por excerto. Para isto foram implementados mecanismos de separação de fontes, dividindo o áudio original em elementos musicais de voz e acompanhamento, originando três fontes de áudio (original, vocal, acompanhamento). Estas três fontes foram ainda segmentadas em excertos de 30 segundos e convertidas para mono. Esta opção teve como objetivo replicar a configuração típica de MER, em concordância com a literatura que serviu de base. Todos os excertos obtidos são classificados individualmente em uma das quatro classes, utilizando máquinas de vetores de suporte (SVMs) treinados com as caraterísticas dos áudios, extraídas através da ferramenta Essentia. As caraterísticas musicais que foram utilizadas variam entre os classificadores, pois foram utilizados algoritmos de seleção e *ranking* para selecionar as que mais se adequavam para cada caso específico.

Os resultados obtidos pelos classificadores foram satisfatórios, com um F1-score máximo de 70,5% para o áudio, utilizando 380 características, e 66,6% para a letra, com apenas 85 características. Estes valores estão em linha com os valores *baseline* do artigo que nos serviu de base (áudio), onde foram obtidos 67,5% e 71,7% com 70 e 800 características respetivamente. De destacar que aqui foi usada apenas uma *framework* áudio, com uma eficiência computacional muito superior às académicas. Como trabalho futuro, seria interessante testar características adicionais, mais relevantes em termos musicológicos, ou ainda abordagens recentes como aprendizagem profunda.

Todas estas as funcionalidades foram desenvolvidas em pequenos serviços isolados e independentes, seguindo uma arquitetura de microsserviços, criando um sistema mais tolerante a falhas, escalável e robusto. Estes serviços serão posteriormente integrados num trabalho a ocorrer em paralelo cujo foco é a orquestração de serviços da solução final de MER.

Palavras-chave: reconhecimento de emoção em música, emoções, áudio, lírica, separação de fontes, aprendizagem computacional, microsserviços.

ABSTRACT

Music Emotion Recognition (MER) is a recent research field, part of Music Information Retrieval (MIR), which has been gaining attention over the years. Music is present in human history since the beginning, used in diverse situations, from entertainment to very different areas such as in medicine. It is generally accepted that music is intimately connected to emotions, manifesting themselves in different ways on a physical and psychological level. Although the mechanisms that govern music and emotion are still to be properly understood, researchers have been working to uncover these relations and, as a result, some have proposed proofs-of-concept to demonstrate the possibilities of automatically extracting emotions from music.

The process of identifying emotions in music using automated approaches is complex, time-consuming and delicate, connecting various areas of knowledge such as psychology, where emotions are studied, computer science, where all signal processing and classification of emotions is carried out, or even the need to have some general knowledge about music theory. As expected, the number of existing systems that currently provide MER functionalities is minimal, mostly being academic proofs-of-concept and experiments.

In this work, a robust and scalable MER system was developed, capable of recognizing emotions transmitted in music, adapting Russell's emotional model for four classes: happy, tense, sad and calm. Our system was built on a previous project, which studied the application of microservice paradigms to MER. From the initial proof-of-concept, we took mostly the lessons learnt and from a practical perspective, part of the web application, completely changing the MER logic and increasing the system complexity and capabilities.

Our new system integrates several state-of-the-art MER concepts for audio and lyrics, using the YouTube platform as the main data source. One classifier was trained for the lyrics component, which predicts a single emotion for the entire song, using textual features extracted from it. Four distinct classifiers were developed for the audio component, predicting multiple emotions per song, still returning one single class for each audio clip. To this end, we implemented source separation mechanisms, splitting the original audio into

voice and accompaniment elements, originating three audio sources (original, vocal, accompaniment). Each of these were then segmented into 30-second excerpts and converted to mono. The rationale behind this decision is to replicate as much as possible the standard MER procedure according to the literature. All excerpts are individually classified into one of the four classes using Support Vector Machines (SVMs), trained with audio features extracted with the Essentia audio framework. The exact features that were used are different for each classifier, since feature selection and ranking algorithms were used to select the best ones for each specific case.

The obtained experimental results were satisfactory, with a maximum F1-score of 70,5% for the audio, using 380 features, and 66,6% for the lyrics, with only 85 features. These values are in line with the baseline values presented by the original paper we built on, where 67,5% and 71,7%, with 70 and 800 audio features respectively, were reported. We highlight that, in our case, a single audio framework with a focus on computational performance was used. In the future, it would be interesting to complement this with additional emotionally relevant features, or even deep learning approaches.

All these functionalities were developed in small isolated and independent services, following a microservices architecture, creating a fault-tolerant, scalable and robust system. The services mentioned above will be integrated into the final system, which is part of an ongoing project focused on the service and container orchestration for MER.

Keywords: music emotion recognition, emotions, audio, lyrics, source separation, machine learning, microservices.

AGRADECIMENTOS

Termino assim mais uma etapa no meu percurso académico. Esta foi uma etapa marcada pela adaptação e inovação, em função dos desafios impostos pelas atuais circunstâncias.

A realização deste trabalho só foi possível com o apoio, amor e disponibilidade da minha mãe Irene, do meu pai Hélder e do meu irmão Ricardo, e por isso dedico-lhes este trabalho. Eles foram o meu suporte neste ciclo, privando-se várias vezes de aproveitar certas regalias para me dar esta oportunidade. O meu mais sincero agradecimento.

Um especial obrigado ao Professor Doutor Renato Panda por me acompanhar neste ano. Foi uma pessoa impecável, fazendo sempre o melhor possível para desbloquear possíveis problemas, incentivar-me a criar métodos de trabalho, partilhou o seu conhecimento, procurou e aprendeu diversos tópicos para me ajudar a compreender as matérias e disponibilizou muito do seu tempo, incluindo fins-de-semana, para discutirmos sobre o trabalho.

Agradeço também aos professores do Mestrado em Engenharia Informática do Instituto Politécnico de Tomar pela disponibilidade que sempre demonstraram, por partilharem o seu conhecimento, pela sua dedicação e por se revelarem exemplos a seguir que com certeza me ajudaram a formar não só a nível académico, mas como pessoa.

Agradeço ainda aos meus colegas de Mestrado que tornaram as aulas que se realizavam numa hora tardia muito mais alegres e interessantes. Entre eles agradeço principalmente ao meu colega João Canoso que me acompanhou neste trabalho, mostrando-se sempre disponível para me ajudar no necessário.

Por fim agradeço ao Centro de Informática e Sistemas da Universidade de Coimbra (CISUC), pela disponibilização de infraestrutura computacional, e ao Centro de Investigação em Cidades Inteligentes (Ci2) do Instituto Politécnico de Tomar, financiado pela Fundação para a Ciência e a Tecnologia (UIDP/05567/2020).

ÍNDICE

RESUMO iii

ABSTRACT v

AGRADECIMENTOS vii

ÍNDICE ix

ÍNDICE DE FIGURAS xiii

ÍNDICE DE TABELAS xvii

GLOSSÁRIO xix

Capítulo 1 Introdução 1

1.1. Motivação 1

1.2. Trabalho desenvolvido 2

1.2.1. Solução inicial (prova-de-conceito) 3

1.2.2. Solução desenvolvida 6

1.3. Organização do relatório 10

Capítulo 2 Estado da arte 11

2.1. Panorama atual 11

2.2. Plataformas de *streaming* 12

2.2.1. Spotify 12

2.2.2. Tidal 13

2.2.3. Deezer 13

2.2.4. YouTube Music 14
2.2.5. Considerações finais sobre plataformas de *streaming* 14
2.3. Taxonomias da emoção 15
2.3.1. Conceptualização categórica 15
2.3.2. Conceptualização dimensional 17
2.3.3. Considerações finais 17
2.4. Abordagem típica dos sistemas de MER 18
2.5. *Dataset* 22
2.5.1. Áudio 22
2.5.2. Lírica 23
2.6. Extração de caraterísticas 23
2.6.1. *Frameworks* de áudio 24
2.6.2. Lírica 27
2.7. Algoritmos de aprendizagem computacional 28
2.7.1. *Frameworks* para implementação de SVM 35
2.8. Separação de fontes (*source separation*) 37
2.9. Obtenção de lírica 41
Capítulo 3 Classificação de emoção em música 45
3.1. Preparação dos conjuntos de dados 45
3.2. Extração de caraterísticas 46

3.3. Tratamento de dados inválidos 47
3.4. Eliminação de caraterísticas 50
3.5. Seleção de caraterísticas 56
3.6. Aprendizagem computacional 57
3.7. Resultados 66
3.7.1. Criação dos modelos de classificação para produção 71
Capítulo 4 Planeamento da aplicação final 75
4.1. Arquitetura de microsserviços 76
4.2. Comunicação entre microsserviços 78
4.3. Linguagens de programação utilizadas 80
4.4. Manipulação do áudio 80
Capítulo 5 Desenvolvimento da aplicação final 83
5.1. Comunicação entre microsserviços 85
5.2. Aplicação *web* 87
5.3. Manager 88
5.4. VidExtractor 92
5.5. GenreFinder 94
5.6. LyricsExtractor 97
5.7. LyricsFeaturesExtractor 98
5.8. SourceSeparation 100

5.9. Segmentation 102

5.10. AudioFeaturesExtractor 104

5.11. Classifier 106

Capítulo 6 Conclusão 109

6.1. Trabalho futuro 110

Referências 111

ÍNDICE DE FIGURAS

Figura 1 - Arquitetura da solução inicial 4

Figura 2 - Arquitetura geral do sistema em desenvolvimento 9

Figura 3 - Sequência de ações lógicas realizadas pelo sistema para classificação de músicas 9

Figura 4 - Expressões faciais que representam o conjunto de emoções básicas proposto por Ekman 16

Figura 5 - Círculo de adjetivos de Hevner 16

Figura 6 – Representação de emoções no plano bidimensional proposto por Russell 17

Figura 7 - Abordagem típica de aprendizagem computacional supervisionada aplicada em MER 19

Figura 8 - Possível diagrama para separação de fontes num sistema de MER 21

Figura 9 – Gráfico do tempo de computação necessário para a extração de caraterísticas de cada *framework* 27

Figura 10 - K-Nearest Neighbors 30

Figura 11 - Ilustração do princípio de SVMs 31

Figura 12 - Três possíveis classificadores lineares 32

Figura 13 - Limite de decisão de um classificador SVM 32

Figura 14 - Árvore de decisão 33

Figura 15 - Dados em falta para caraterísticas do componente de acompanhamento 48

Figura 16 - Dados em falta para caraterísticas do áudio original 48

Figura 17 - Dados em falta para caraterísticas da componente vocal 49

Figura 18 - Dados em falta para caraterísticas da lírica 49

Figura 19 - Excerto de uma matriz de correlação entre pares de características extraídas do áudio 52

Figura 20 - Distribuição das caraterísticas por tipo e por fonte..... 53

Figura 21 - Caraterísticas de baixo nível discriminadas por grupos de caraterísticas e fonte musical..... 54

Figura 22 - Caraterísticas de ritmo discriminadas por grupos de caraterísticas e fonte musical 55

Figura 23 - Caraterísticas de tonal discriminadas por grupos de caraterísticas e fonte musical 56

Figura 24 - Parâmetros testados para classificadores 58

Figura 25 - Desempenho dos *kernels* para as caraterísticas de acompanhamento 59

Figura 26 - Desempenho dos *kernels* para todas as caraterísticas de áudio combinadas 60

Figura 27 - Desempenho dos *kernels* para as características do áudio original 60

Figura 28 - Desempenho dos *kernels* para as caraterísticas vocais 60

Figura 29 - Desempenho dos *kernels* para as caraterísticas da lírica 61

Figura 30 - Desempenho dos melhores parâmetros do *kernel* poly para caraterísticas de acompanhamento..... 61

Figura 31 - Desempenho dos melhores parâmetros do *kernel* rbf para todas as caraterísticas de áudio 62

Figura 32 - Desempenho dos melhores parâmetros do *kernel* rbf para caraterísticas do áudio original..... 62

Figura 33 - Desempenho dos melhores parâmetros do *kernel* poly para caraterísticas da voz 62

Figura 34 - Desempenho dos melhores parâmetros do *kernel* sigmoid para caraterísticas da lírica 63

Figura 35 - Métrica F1-score para todos os modelos treinados das caraterísticas de acompanhamento 64

Figura 36 - Métrica F1-score para todos os modelos treinados das caraterísticas de áudio combinadas 64

Figura 37 - Métrica F1-score para todos os modelos treinados das caraterísticas do áudio original 65

Figura 38 - Métrica F1-score para todos os modelos treinados das caraterísticas da voz ... 65

Figura 39 - Métrica F1-score para todos os modelos treinados das caraterísticas da lírica 66

Figura 40 - Distribuição das melhores 400 características (top400) áudio do conjunto das 3 fontes combinadas, distribuídas por grupo e tipo 71

Figura 41 - Matriz de confusão obtida para o classificador de acompanhamento (90% treino, 10% teste) 73

Figura 42 - Exemplo de arquitetura de microsserviços que utiliza filas de mensagens 76

Figura 43 – Estrutura de 3 aplicações *containerizadas* no mesmo sistema..... 77

Figura 44 - Visão geral do protocolo AMQP 0-9-1 79

Figura 45 - Arquitetura geral da comunicação entre microsserviços na solução implementada 85

Figura 46 - Filas de mensagens implementadas 86

Figura 47 - Exemplo de estrutura das mensagens 87

Figura 48 - Estrutura dos documentos da base de dados NoSQL 90

Figura 49 - Lógica do microsserviço vidExtractor 94

Figura 50 - Lógica do microsserviço GenreFinder 96

Figura 51 - Lógica do microsserviço LyricsExtractor 98

Figura 52 - Lógica do microsserviço LyricsFeaturesExtractor 99

Figura 53 - Lógica do microsserviço SourceSeparation 101

Figura 54 - Exemplo de segmentação para áudio de 50 segundos 102

Figura 55 - Lógica do microsserviço Segmentation 104

Figura 56 - Lógica do microsserviço AudioFeaturesExtractor 106

Figura 57 - Lógica do microsserviço Classifier 107

ÍNDICE DE TABELAS

Tabela 1 - Comparação das diferentes ferramentas de extração de características 27

Tabela 2 - Instâncias com soluções pretendidas 29

Tabela 3 - Comparação entre algoritmos de aprendizagem automática para problemas genéricos segundo (Kotsiantis, 2007) 34

Tabela 4 - Tempo de processamento para 2 *stems* 39

Tabela 5 - Tempo de processamento para 4 *stems* 39

Tabela 6 - Melhores combinações de parâmetros para cada dataset e algumas métricas ... 69

Tabela 7 - Tabela de mapeamento entre microsserviços e filas de mensagens 87

GLOSSÁRIO

API	*Application Programming Interface* - Conjunto de funções e procedimentos que permitem criar aplicações para serem acedidas por serviços externos
Broker	Um *broker*, quando associado ao paradigma de fila de mensagens, serve de intermediário. Recebe uma mensagem de um transmissor e envia-a para um ou mais recetores.
Copyright	É um tipo de propriedade que dá ao seu dono o direito exclusivo sobre uma obra
CPU	*Central Processing Unit* – Circuito eletrónico, também conhecido como processador, que realiza as instruções de um programa de computador
Framework	Fornece funcionalidades genéricas para auxiliar o desenvolvimento de *Software*
GPU	*Graphics Processing Unit* – Circuito eletrónico projetado para manipular os gráficos do computador e processamento de imagem
Playlist	Lista de ficheiros de vídeo ou áudio para serem reproduzidos
Software	Coleção de instruções, dados ou programas que informam o computador como executar tarefas específicas
Streaming	Disponibilização de conteúdo, por exemplo multimédia, através da Internet de forma contínua, permitindo a visualização do mesmo antes de todo o conteúdo ser recebido.
Threads	Um processo pode dividir-se em múltiplas tarefas que podem ser executadas em paralelo. Estas tarefas chamam-se *threads*.

Capítulo 1
Introdução

A música acompanha a história e a evolução do ser humano há milhares de anos. Está presente em todas as sociedades conhecidas, desde a mais primitiva até à mais avançada, e ainda no universo animal, apesar dos sons produzidos pelos animais nem sempre parecerem melódicos ao ouvido do ser humano.

Existem várias áreas onde a música pode ser aplicada. Pode ser simplesmente utilizada para entretenimento e diversão ou para situações mais sérias e complexas como tratamentos na área da saúde. Tendo em conta uma das mais recentes crises mundiais, a pandemia Covid-19, que "… está a gerar *stress* em toda a população" (World Health Organization, 2020), a música pode mostrar-se eficiente na prevenção deste sentimento que pode originar problemas físicos e emocionais porque "ouvir música está altamente associado à redução do *stress*…" (de Witte et al., 2020). Isto acontece porque a música transmite emoções (Juslin, 2013), despertando sinais positivos ou negativos no ouvinte. Para ajudar, "nunca antes houve uma coleção de músicas tão grande a ser criada e acedida diariamente" (Yang & Chen, 2012), isto porque a popularidade da Internet e a facilidade de aceder à mesma impulsiona este movimento.

1.1. Motivação

Os métodos de distribuição de música sofreram várias alterações ao longo dos anos. Todos tiveram importância em determinadas circunstâncias, desde os formatos analógicos até aos mais recentes formatos digitais. Atualmente utiliza-se bastante os serviços de *streaming*, como Spotify[1] ou Tidal[2], que permitem ouvir e descobrir músicas sem a necessidade de as descarregar ou comprar. Por outro lado, a quantidade de músicas disponíveis nestas plataformas é imensa, o que pode tornar o processo de procura ineficiente utilizando os

[1] https://www.spotify.com/us/
[2] https://tidal.com/

mecanismos de pesquisa habituais, que se baseiam em meta-dados definidos manualmente, como o título da música, o nome do artista ou do álbum. Sendo que a música transmite emoções, seria interessante utilizar esta informação, uma caraterística do próprio objeto, no processo de consulta. Existem alguns serviços que permitem pesquisas baseadas em "*moods"* (humores), mas por norma esta informação é etiquetada manualmente. É ainda expectável que, num futuro próximo, fique disponível no Spotify uma nova solução para recomendação de músicas utilizando como base informação extraída da voz do utilizador no momento da pesquisa[3].

Com base nisto, surgiu nas últimas décadas um ramo de investigação específico da área de Recuperação de Informação Musical (MIR, Music Information Retrieval) com o nome de Reconhecimento Emocional em Música (MER, Music Emotion Recognition). Esta é uma área com diversos problemas ainda em aberto, que tem vindo a crescer bastante nos últimos anos com diversos estudos a serem publicados. Tipicamente, uma abordagem MER é constituída por três elementos distintos que serão explicados mais em detalhe no próximo capítulo. Resumidamente, consiste em 1) obter um *dataset*, ou seja, obter um conjunto de músicas e respetivas anotações emocionais, 2) processamento e tratamento do sinal de áudio utilizando algoritmos computacionais para extração de características do *dataset* e 3) utilização de algoritmos de aprendizagem computacional para reconhecer padrões entre as características extraídas e respetivas anotações emocionais nos exemplos do *dataset* e assim prever a emoção de uma música desconhecida.

1.2. Trabalho desenvolvido

O objetivo deste trabalho foi o desenvolvimento de um sistema de classificação emocional em músicas, resiliente e escalável, com uma abordagem e desempenho semelhante ao encontrado no estado de arte da área, transpondo conceitos de investigação fundamental para algo utilizável. Este trabalho teve como ponto de partida uma prova de conceito realizada a nível académico pelo aluno Ricardo António denominada "Microsserviços para

[3] https://visao.sapo.pt/exameinformatica/noticias-ei/internet/2021-01-29-spotify-quer-recomendar-musicas-com-base-nos-sentimentos-dos-utilizadores/

Reconhecimento de Emoção em Música" (António, 2019), onde foi demonstrada a viabilidade da ideia. Nas próximas duas secções é explicado no que consistia esta solução inicial, referidos alguns dos aspetos e melhorias que se podia implementar e, por fim, é apresentada de forma breve a solução proposta, fruto do presente trabalho.

1.2.1. Solução inicial (prova-de-conceito)

A solução inicial (prova de conceito desenvolvida anteriormente) consiste num sistema de MER demonstrável que, essencialmente, permite a análise de emoções a partir de um sinal de áudio de vídeos existentes na plataforma YouTube[4]. A interação entre o utilizador e o sistema é através de uma aplicação *web*.

De forma geral, o procedimento para classificar uma música começa na página inicial da aplicação *web*, onde qualquer utilizador pode inserir o endereço de um vídeo existente no YouTube e fazer o pedido de classificação. O vídeo referente ao endereço inserido é descarregado, é feito o processamento do seu sinal de áudio e é lhe atribuída uma classificação emocional (entre alegre, tensa, triste e calma) utilizando um classificador emocional (algoritmo SVM, descrito na secção 2.7), que foi previamente treinado utilizando um conjunto de dados disponível publicamente. Este *dataset* contém 900 excertos de áudio e respetivas anotações emocionais (descrito na secção 2.5.1).

Posteriormente, o vídeo fica disponível para visualização na aplicação *web*, assim como algumas informações referentes ao mesmo, incluindo a emoção obtida.

O sistema, ilustrado na Figura 1, foi projetado seguindo uma arquitetura de microsserviços (explicado na secção 4.1), onde cada um destes serviços é responsável por uma tarefa específica, estando isolado dos restantes. Os vários serviços comunicam entre si através de um sistema de fila de mensagens. A implementação destes pequenos serviços é realizada utilizando *containers* construídos utilizando a ferramenta Docker, que permite a execução e independência de cada um dos *containers*.

[4] https://www.youtube.com/

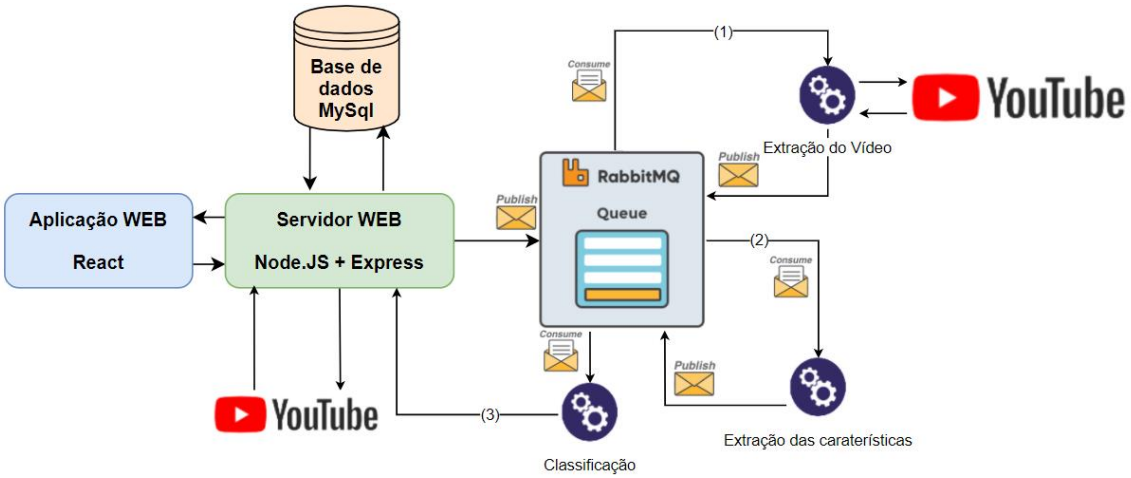


Figura 1 - Arquitetura da solução inicial

A aplicação *web* contém algumas funcionalidades adicionais. Os utilizadores não autenticados conseguem verificar quais as músicas que estão a ser processadas, podem pesquisar as músicas já classificadas através do título dos vídeos, pedir a classificação de novas músicas e visualizar os últimos vídeos classificados. São disponibilizados alguns detalhes como o número de visualizações e gostos de cada vídeo, sempre atualizados no momento da consulta, associados a cada vídeo.

Os utilizadores autenticados têm acesso a mais funcionalidades para além das anteriores. Possuem um perfil onde podem alterar os seus dados, ver a lista de vídeos que pediram para classificar e têm acesso a listas de reprodução. O número de listas é ilimitado, assim como o número de vídeos que estas podem conter. A qualquer momento o utilizador pode adicionar ou remover qualquer vídeo ou lista criados pelo próprio. Podem também avaliar a classificação atribuída a um vídeo.

O utilizador autenticado com permissões de administrador pode ainda listar e criar utilizadores, assim como eliminar qualquer vídeo da plataforma.

Os microsserviços desenvolvidos nesta versão inicial são três, como ilustrado na Figura 1, e comunicam por um intermediário de mensagens. O primeiro microsserviço a ser executado no processo de classificação é o microsserviço de Extração do Vídeo. Inicialmente é feita uma validação para verificar se o vídeo inserido pelo utilizador pertence à categoria das

músicas (informação fornecida pelo YouTube). Se assim se verificar, o mesmo será descarregado e o seu identificador (*id* do vídeo) será colocado na fila de mensagens para ser consumido pelo serviço seguinte.

De seguida, o microsserviço Extração de Caraterísticas recebe a informação de que foi colocado na fila de mensagens o identificador de um vídeo e inicia a extração de três caraterísticas do sinal de áudio referente ao respetivo identificador, para serem utilizadas no processo de previsão da emoção. Após as caraterísticas serem extraídas, são igualmente colocadas na fila de mensagens.

Por fim, o microsserviço Classificação utiliza as caraterísticas extraídas para atribuir uma emoção à nova música, com base num modelo previamente treinado utilizando aprendizagem computacional. No final deste processo, a música tem a sua emoção atualizada na base de dados e fica disponível para visualização na plataforma *web*.

O modelo utilizado para fazer a previsão da emoção foi previamente treinado e testado com um *dataset* (descrito em 2.5.1) de 900 músicas etiquetadas. Para isso foram extraídas as mesmas três caraterísticas de todos os sinais de áudio, que são duas caraterísticas de baixo nível e uma caraterística de ritmo: 1) valor médio da saliência do tom (*pitch salience*), 2) valor médio de ruído (*average loudness*) e 3) número de batimentos por minuto. A previsão de emoções foi feita utilizando máquinas de vetores de suporte (Support Vector Machines, explicado na secção 2.7) com os parâmetros por defeito para identificar padrões entre as caraterísticas e as respetivas etiquetas (emoção).

Esta primeira versão foi desenvolvida como prova de conceito, servindo de estudo sobre a aplicação de conceitos avançados de engenharia de *software* a um problema normalmente ligado à investigação fundamental – extração de informação emocional de sinais áudio. Por essa razão, contém diversas limitações que foram identificadas e parte do foco deste trabalho. Entre estas, a extração de poucas caraterísticas do sinal de áudio (apenas três) pode limitar a taxa de acertos do classificador e, por isso, deve ser extraído um maior número de caraterísticas relevantes. Isto não significa que devem ser extraídas todas as caraterísticas possíveis, ou até podem ser todas extraídas, mas nem todas devem ser utilizadas porque

tendo demasiadas caraterísticas irrelevantes ao tema abordado pode resultar num excesso de informação, aumentando a complexidade do classificador (Hira & Gillies, 2015).

Uma possível melhoria a realizar é fazer a segmentação do sinal de áudio. Sendo que a tarefa de análise e processamento do sinal de áudio já é complexa e pesada, a utilização do sinal de áudio completo para a extração de caraterísticas exige ainda mais recursos computacionais que podem ser reduzidos segmentando o sinal de áudio. Para além da afirmação anterior, é também um facto que a emoção transmitida numa música nem sempre é a mesma ao longo desta. É habitual que exista mais do que uma única emoção durante uma música e, com o áudio segmentado, é possível prever a emoção de cada um dos excertos.

Outras pontos que foram explorados incluem a experimentação de diferentes parâmetros de configuração do classificador e a classificação de áudio separando o áudio completo em voz e acompanhamento, uma vez que isto pode levar a um aumento de desempenho (R. E. S. Panda, 2019). Do ponto de vista de orquestração e resiliência, estava versão inicial tinha uma série de problemas para as quais foram propostas soluções, sendo este o foco do trabalho do João Canoso, pelo que os detalhes estarão descritos no seu trabalho de mestrado.

1.2.2. Solução desenvolvida

A solução proposta segue o paradigma de arquitetura testado na prova de conceito anterior. Poderia ser utilizada uma arquitetura monolítica, ou seja, o sistema ser composto por uma "peça" única, mas uma arquitetura de microsserviços adequa-se melhor ao objetivo do projeto, tal como foi concluído no trabalho anterior. Este tipo de arquitetura permite que os módulos sejam independentes entre si, logo se algum destes módulos falhar, o sistema continua a funcionar exceto o microsserviço que falhou, podendo ser lançado novamente no mesmo ou noutro nó de computação se for bem configurado. Permite ainda que todo o processo possa ser realizado paralelamente, em vez de ter o sistema "parado" à espera que uma música seja classificada para poder iniciar o processo novamente para uma nova música, onde é possível ter réplicas dos microsserviços, escalando a carga conforme necessário. Os microsserviços comunicam entre si através de filas de mensagens. Toda a componente de gestão e orquestração dos microsserviços, mecanismos de tolerância a falhas, entre outras funcionalidades é o foco do trabalho de mestrado em desenvolvimento pelo

aluno João Canoso, também do Instituto Politécnico de Tomar, pelo que será detalhada nesse trabalho. Este relatório é essencialmente focado no estudo de mecanismos de classificação de emoção em música com base no estado de arte da área e o desenvolvimento dos microsserviços para obter a classificação de emoções num sistema real.

Os microsserviços desenvolvidos[5] estão identificados na Figura 2 pelos ícones a verde. Como é também possível verificar, estes microsserviços comunicam entre si utilizando as várias filas de mensagens ilustradas a laranja, através de um *broker*. O paradigma utilizado é de produtor/consumidor, onde os microsserviços publicam mensagens nas filas para serem consumidas apenas uma única vez por outros.

Para tornar os microsserviços mais independentes, de forma que eles realmente não comuniquem entre si e nem necessitem de saber da existência dos outros, é definido um microsserviço nomeado por Manager onde é definida toda sequência lógica de instruções do sistema, ou seja, a forma como o sistema se comporta a nível de interação entre os diversos microsserviços para classificação emocional é gerida por este, que pode também ele ser paralelizado. Assim, os microsserviços quando terminam as suas tarefas, colocam as mensagens na fila *Management* que o Manager escuta e identifica qual o próximo passo para aquele registo.

A sequência de ações definida no Manager está ilustrada na Figura 3, onde cada componente será um microsserviço e fará apenas as tarefas a ele destinadas. De forma semelhante à prova de conceito anterior, também neste sistema as músicas a classificar são do YouTube e o processo de classificação tem início na aplicação *web* onde o utilizador insere o endereço de um vídeo do YouTube. Quando um utilizador deseja obter a classificação de uma música, será consultada a base de dados para avaliar se a mesma já foi classificada anteriormente e, apenas no caso de não ter sido ainda classificada, é iniciado o processo de classificação. Para tal, é colocada na fila de mensagens a indicação que se pretende classificar uma nova música. A categoria da mesma é avaliada para verificar que se trata realmente de uma música e não

[5] Alguns microsserviços, como o de *logger*, estão ainda em desenvolvimento e dependentes do trabalho do João Canoso

de um vídeo generalista (não musical). De seguida é descarregada e tem os seus meta-dados extraídos e guardados numa base de dados não-relacional, NoSQL (Not Only SQL). Neste momento, assim como mostra a Figura 3, algumas instruções podem ser realizadas em paralelo, visto que não causam impacto nas outras, são independentes, dividindo-se em três ramos. No ramo da esquerda será tratada e computada a componente da lírica, onde através dos meta-dados extraídos, essencialmente o título da música e nome do artista, se pretende obter a lírica da música. Em seguida serão extraídas algumas caraterísticas relevantes da lírica (processamento de texto) por forma a, através de um modelo corretamente treinado e testado, obter uma classificação emocional para a mesma.

No ramo do meio é tratada e processada a componente do áudio. São identificadas e separadas as fontes/elementos da música, podendo ser a voz, guitarra, piano, entre outros. No sistema atual, é feita a separação entre voz e o acompanhamento. Em seguida, os ficheiros de áudio dos vários elementos são segmentados em excertos de tamanho fixo e é realizado um procedimento de *downsampling*, que consiste em diminuir o número de amostras, convertendo o áudio para *mono channel* (o áudio é transmitido através de um único canal) e o valor de taxa de amostragem diminuído para 22,05kHz (*sampling rate* – um microfone regista 22 500 amostras, valores de pressão, por segundo). Este passo é feito para aligeirar o processamento do áudio a níveis computacionais, sendo que o número de amostras é reduzido significativamente porque o habitual é a utilização de dois canais (stereo) a 44,1 kHz. Seguidamente são extraídas as caraterísticas dos vários excertos e, através de alguns modelos de classificação previamente treinados e testados, é identificada a emoção para cada excerto.

No ramo da direita é obtido o género musical (por exemplo rock, pop, hip-hop, entre outros) que futuramente poderá vir a ser também utilizado como uma caraterística dos modelos de aprendizagem automática.

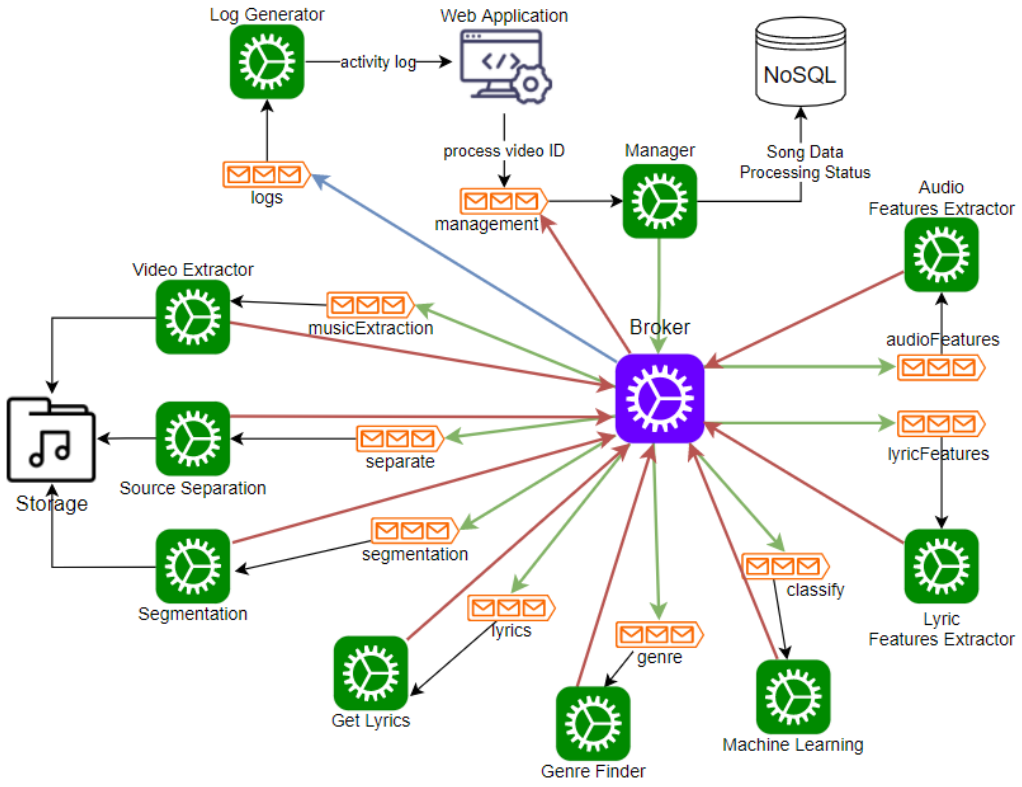


Figura 2 - Arquitetura geral do sistema em desenvolvimento

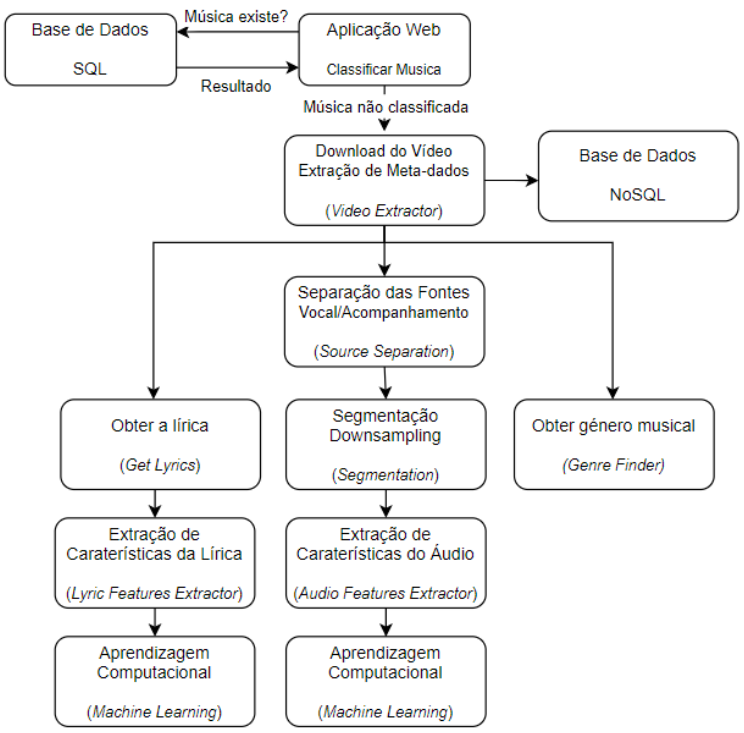


Figura 3 - Sequência de ações lógicas realizadas pelo sistema para classificação de músicas

1.3. Organização do relatório

Este relatório está organizado da seguinte forma. No capítulo 1 foi dada uma visão geral do trabalho realizado, explicando em que consiste o problema do reconhecimento de emoções em música, a motivação e de forma breve a solução desenvolvida. No capítulo 2 é abordado o estado de arte na área, incluindo o processo típico de reconhecimento emocional em música e também a análise das várias tecnologias existentes necessárias para atingir o objetivo proposto no nosso sistema. O capítulo 3, mais focado na parte de investigação, aborda todo o estudo e criação dos modelos de aprendizagem automática para reconhecimento de emoções. O capítulo 4 apresenta uma componente de planeamento, mais a nível de arquitetura geral, tendo por base o estudo prévio feito no estado de arte e na prova de conceito anterior. No capítulo 5 é descrito detalhadamente o sistema desenvolvido, incluindo a engenharia dos vários microsserviços. Por fim, o capítulo 6 finaliza o trabalho, apresentando as conclusões obtidas e possíveis melhorias para um trabalho futuro.

Capítulo 2
Estado da arte

Este capítulo apresenta uma revisão do estado de arte relacionado com cada um dos blocos identificados na solução proposta, sendo o foco a área do reconhecimento emocional em música. A secção 2.1 descreve o panorama atual nesta área de MER e a indústria musical. Segue-se a secção 2.2 onde são apresentadas algumas das mais conhecidas plataformas de *streaming* de música e é escolhida a fonte de dados que irá servir o sistema. As secções 2.3 e 2.4 descrevem respetivamente uma componente teórica do reconhecimento emocional em música, nomeadamente algumas das taxonomias de emoção que existem, sem entrar muito detalhadamente na parte de psicologia e a abordagem típica dos sistemas MER. As restantes secções apresentam algumas tecnologias estudadas e respetivas conclusões e justificações sobre a utilização das mesmas.

2.1. Panorama atual

O reconhecimento de emoções em músicas é relativamente recente, faz parte do campo recuperação de informações em músicas e tem recebido uma atenção crescente nos últimos anos. Determinar o conteúdo emocional de um sinal de áudio computacionalmente é um processo complexo porque para além da necessidade de compreender o processamento de um sinal de áudio e a aprendizagem computacional, são também necessários alguns conhecimentos de perceção auditiva, psicologia e teoria musical (Kim et al., 2010).

A indústria de distribuição de música tem crescido imensamente ao longo dos anos. Como descrito anteriormente, a cada dia que passa existe cada vez mais músicas a serem adicionadas em plataformas de *streaming*. Este evento tem despertado o interesse da indústria em desenvolver novos mecanismos de pesquisa para ajudar na catalogação das bases de dados musicais, em alternativa aos mecanismos já habituais baseados em meta-dados definidos manualmente, como o título da música. “Em particular, métodos baseados no conteúdo emocional de uma música têm um papel considerável, fornecendo métodos muito mais refinados para uma melhor navegação e filtragem de tais bases de dados” (R. E. S. Panda, 2019).

2.2. Plataformas de *streaming*

"Plataformas de *Streaming* de música oferecem serviços que permitem aos utilizadores ouvir áudios, *podcasts* ou ver vídeos de músicas"[6] a partir de qualquer lugar através de dispositivos ligados à Internet. As várias plataformas existentes baseiam-se bastante nos meta-dados das músicas para a sua consulta e organização, existindo apenas algumas plataformas que oferecem outros recursos como recomendação de músicas ou personalização automática de listas de reprodução, sendo muito incomum ou até inexistente a catalogação de músicas pela emoção transmitida. De seguida são apresentados alguns exemplos de plataformas de *streaming* de música.

2.2.1. Spotify

Spotify[7] disponibiliza milhões de músicas e *podcasts*. Permite navegar pelas coleções de músicas de amigos, artistas e celebridades, ou criar estações de rádio[8]. A consulta de músicas e *podcasts* baseia-se fundamentalmente nos seus meta-dados, como o nome de artistas, álbuns ou géneros musicais, mas também são disponibilizadas *playlists* com as melhores músicas de cada país por exemplo[9]. Conta ainda com mecanismos de sugestão de músicas, recomendando músicas ao utilizador com base nos seus padrões de pesquisa.

A plataforma é de fácil acesso, podendo ser acedida através do navegador (*browser*) ou através de uma aplicação que pode ser executada em *smartphones*, computadores, *tablets*, entre outros dispositivos. Os serviços descritos anteriormente fazem parte do plano grátis, havendo ainda subscrições pagas que disponibilizam outras funcionalidades como o *download* de músicas ou melhoria na qualidade das transmissões.

[6] https://www.grandviewresearch.com/industry-analysis/music-streaming-market
[7] https://www.spotify.com/pt-en/
[8] https://www.spotify.com/us/about-us/contact/
[9] https://open.spotify.com/search

2.2.2. Tidal

Tidal[10] é uma plataforma de *streaming* de música e entretenimento focada na qualidade do som para ter uma ligação mais profunda com a arte. Está disponível um pouco por toda a parte do globo, em cerca de 56 países, e conta com mais de 70 milhões de músicas dos vários géneros musicais conhecidos e 250 mil vídeos de alta qualidade[11]. Permite criar listas de reprodução e consultar outras listas de reprodução de artistas ou editores de música. A consulta de músicas pode ser feita através do nome das músicas, vídeos, artistas, álbuns, géneros musicais ou listas de reprodução. Os utilizadores têm ainda a possibilidade de participar em concertos com artistas consagrados ou emergentes, e outras interações diretas entre fãs e os artistas favoritos. A plataforma não disponibiliza um plano grátis e, por isso, o acesso e todos os vários recursos descritos acima estão apenas disponíveis através de subscrições pagas.

2.2.3. Deezer

Deezer[12] conta com mais de 16 milhões de utilizadores ativos, cerca de 56 milhões de temas, entre músicas novas e antigas, e é acedível em mais de 180 países[13] através do navegador ou da aplicação, quer seja para computador ou dispositivos móveis. Tem diferentes tipos de subscrição[14], sendo a subscrição gratuita mais limitada, permitindo a consulta de músicas através de listas de reprodução, álbuns, artistas, *podcasts*, canais, perfis e géneros musicais. Este tipo de subscrição permite ainda que o utilizador receba sugestões de músicas de editoras discográficas parceiras e sugestões com base num mecanismo que aprende com os gostos do utilizador baseado nas músicas que este acompanha. As subscrições pagas têm algumas outras caraterísticas, destacando-se o facto de o utilizador poder descarregar as músicas, utilizando a aplicação em modo *offline*.

[10] https://tidal.com/
[11] https://tidal.com/whatistidal
[12] https://www.deezer.com/pt/
[13] https://www.deezer.com/pt/company/press
[14] https://www.deezer.com/pt/offers

2.2.4. YouTube Music

YouTube Music é uma aplicação desenvolvida pela Google focada no *streaming* de músicas que existem na plataforma oficial do YouTube. É uma aplicação multiplataforma que pode ser acedida em qualquer lugar sem acesso à internet através de dispositivos móveis e *desktop*, disponibilizando vários recursos, entre eles pesquisas de música por álbuns, artistas, listas de reprodução, tendências, novos lançamentos, atuações ao vivo, *remixes*, géneros e estado de espírito[15]. Permite ainda a pesquisa de músicas através da letra musical e descrição da música, fazendo uso de uma tecnologia de inteligência artificial[16]. Estes recursos podem ser acedidos gratuitamente ou através de subscrições pagas.

2.2.5. Considerações finais sobre plataformas de *streaming*

As plataformas de *streaming* de música são benéficas na vida das pessoas, sendo acedidas diariamente por milhares de utilizadores para consumir um número de músicas imensurável. A tendência é que o número de músicas adicionadas diariamente a estas plataformas continue a crescer porque todos os dias são produzidas e disponibilizadas bastantes músicas. O principal objetivo destas aplicações comerciais é disponibilizar músicas e vídeos de alta qualidade, permitindo na sua maioria a consulta de músicas através dos seus meta-dados. Por vezes são implementados mecanismos diferentes, mas pouco inovadores. Há ainda casos em que para além da utilização do género musical, são utilizados "*moods*" para sugerir músicas, mas estas categorias são igualmente definidas manualmente.

Assim, com o aumento do número de músicas disponíveis e com a, por vezes, ineficiência dos habituais mecanismos de pesquisa, é expectável que exista uma procura por novos mecanismos, que é o que se pretende com este trabalho, a consulta de músicas utilizando uma característica do próprio objeto, a sua emoção. Este mecanismo pretende basear-se no

[15] https://www.youtube.com/musicpremium
[16] https://pt.wikipedia.org/wiki/YouTube_Music

processamento do sinal de áudio e lírica para disponibilizar informações emocionais e permitir a catalogação através destas informações.

A plataforma do YouTube foi a fonte de dados escolhida para utilizar neste sistema porque não é necessário criar uma conta para obter os vídeos e é totalmente aberta, tornando-se assim na plataforma que melhor facilitaria no processo de obter músicas.

2.3. Taxonomias da emoção

Reconhecer emoções é um processo natural para o ser humano. Mas na verdade, as emoções são um tópico subjetivo. Isto deve-se ao facto de as emoções variarem de pessoa para pessoa, de momento para momento e mesmo entre culturas. As pessoas podem ter perceções diferentes dos mesmos estímulos e por isso utilizarem palavras diferentes para o descreverem, como sinónimos ou pequenas variações.

O passo inicial para qualquer sistema de MER é a escolha de qual a taxonomia a utilizar porque os resultados obtidos do processo de classificação serão representados segundo a taxonomia escolhida. Várias taxonomias têm sido propostas na área da psicologia, agrupando-se nas duas conceptualizações apresentadas de seguida.

2.3.1. Conceptualização categórica

A conceptualização categórica utiliza palavras ou conjuntos de palavras para descrever emoções. Fazem parte desta abordagem diversos modelos distintos, como por exemplo o conjunto de emoções básicas proposto por Ekman[17], onde são utilizadas seis palavras para descrever as emoções associadas a expressões faciais, representadas na Figura 4, e são elas: felicidade, tristeza, medo, nojo, raiva e surpresa. Outro modelo bastante conhecido é o círculo de adjetivos de Hevner (*Hevner's adjective circle or clock*) criado por Kate Hevner[18]. Este modelo é um conjunto de grupos de adjetivos, dispostos em círculo, onde cada grupo

[17] https://en.wikipedia.org/wiki/Paul_Ekman
[18] https://en.wikipedia.org/wiki/Kate_Hevner_Mueller

contém adjetivos com significados semelhantes utilizados para descrever os estados emocionais, como ilustra a Figura 5.

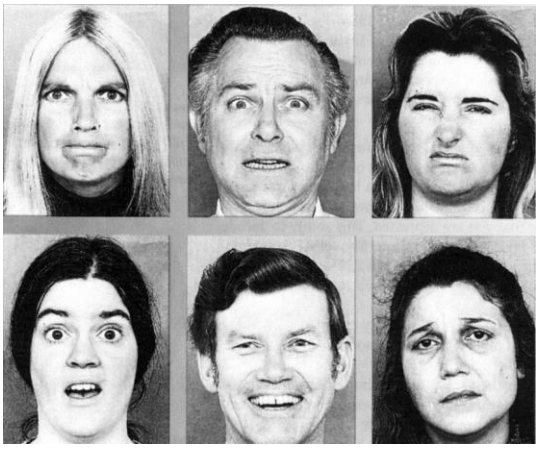


Figura 4 - Expressões faciais que representam o conjunto de emoções básicas proposto por Ekman[19]

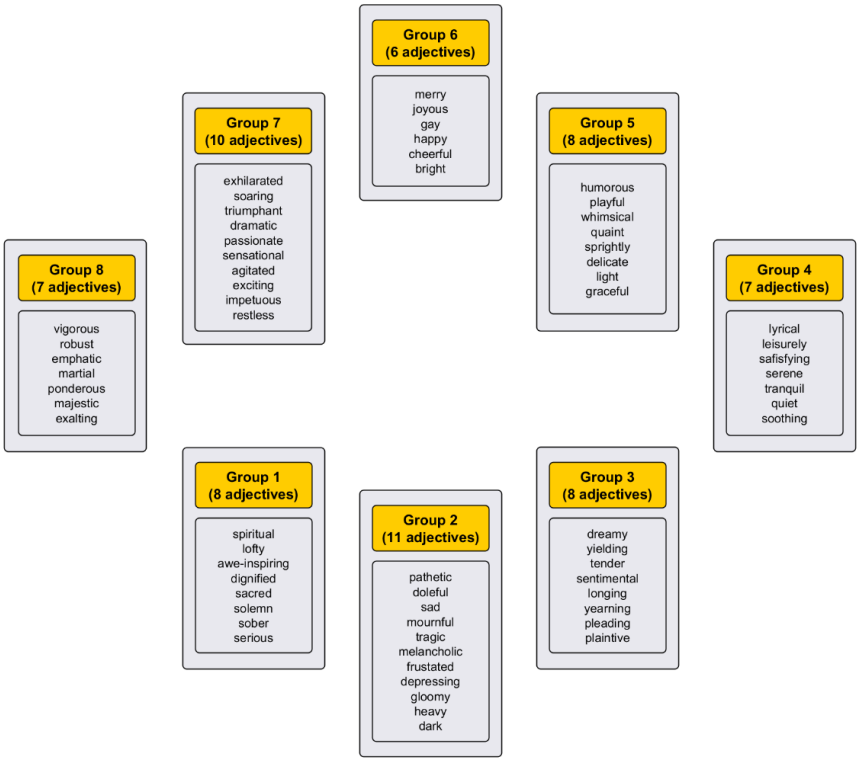


Figura 5 - Círculo de adjetivos de Hevner[20]

[19] Figura 2.3, retirada da página 25 de (R. E. S. Panda, 2019)
[20] Figura 2.4, retirada da página 27 de (R. E. S. Panda, 2019)

2.3.2. Conceptualização dimensional

A conceptualização dimensional pretende identificar emoções baseando-se em descritores representados em planos dimensionais. Um dos modelos mais conhecidos é o *Russell's Circumplex Model of Emotion*, representado na Figura 6, que admite que as emoções existem num plano bidimensional de quatro quadrantes ao longo dos eixos independentes de excitação (*arousal*) e valência (*valence*) (Kim et al., 2010). As várias emoções podem ser aproximadamente definidas no primeiro quadrante como felizes ou energéticas, no segundo encontram-se emoções melancólicas ou agressivas, no terceiro as emoções depressivas e no quatro quadrante estão as emoções calmas.

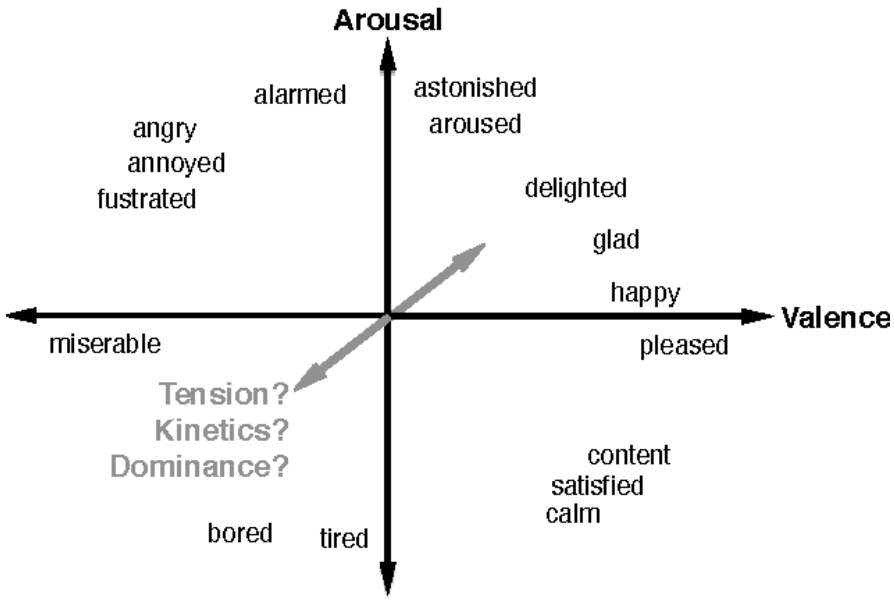


Figura 6 – Representação de emoções no plano bidimensional proposto por Russell[21]

2.3.3. Considerações finais

Existem vários modelos propostos para ambas as conceptualizações categóricas e dimensionais, sendo óbvio que ambas apresentam vantagens e desvantagens. Para a conceptualização categórica é vantajoso o facto de que o ser humano está habituado a atribuir etiquetas discretas, palavras, para descrever emoções. Por outro lado, tomando a

[21] Figura 1, retirada da página 257 de (Kim et al., 2010)

categorização de músicas como exemplo, estes modelos são incapazes de distinguir músicas da mesma categoria, ou seja, admitindo um conjunto de músicas classificadas como alegres, torna-se impossível averiguar se uma música é mais alegre do que outra, são igualmente alegres e por isso iguais para efeitos de pesquisa.

Esta desvantagem dos modelos categóricos é solucionada pelos modelos dimensionais porque como as emoções são representadas num plano dimensional, quer seja bidimensional ou tridimensional, cada ponto no plano corresponde a uma emoção diferente. Possíveis desvantagens identificadas são o nível elevado de complexidade computacional e até mesmo para o ser humano, que não está habituado a utilizar descritores como valência e excitação para definir as emoções.

Em suma, para este trabalho será seguida a conceptualização categórica, utilizando um modelo simples de quatro classes sendo estas: Alegre, Tensa, Triste e Calma, representando os quadrantes 1, 2, 3 e 4 respetivamente.

2.4. Abordagem típica dos sistemas de MER

A abordagem típica de *machine learning* aplicada a soluções MER baseia-se em três elementos distintos, tal como ilustrado na Figura 7:

1. Obter um conjunto de dados verdadeiros etiquetados emocionalmente (*dataset*);
2. Análise e processamento do sinal de áudio;
3. Aprendizagem computacional (fases de treino, teste e classificação)

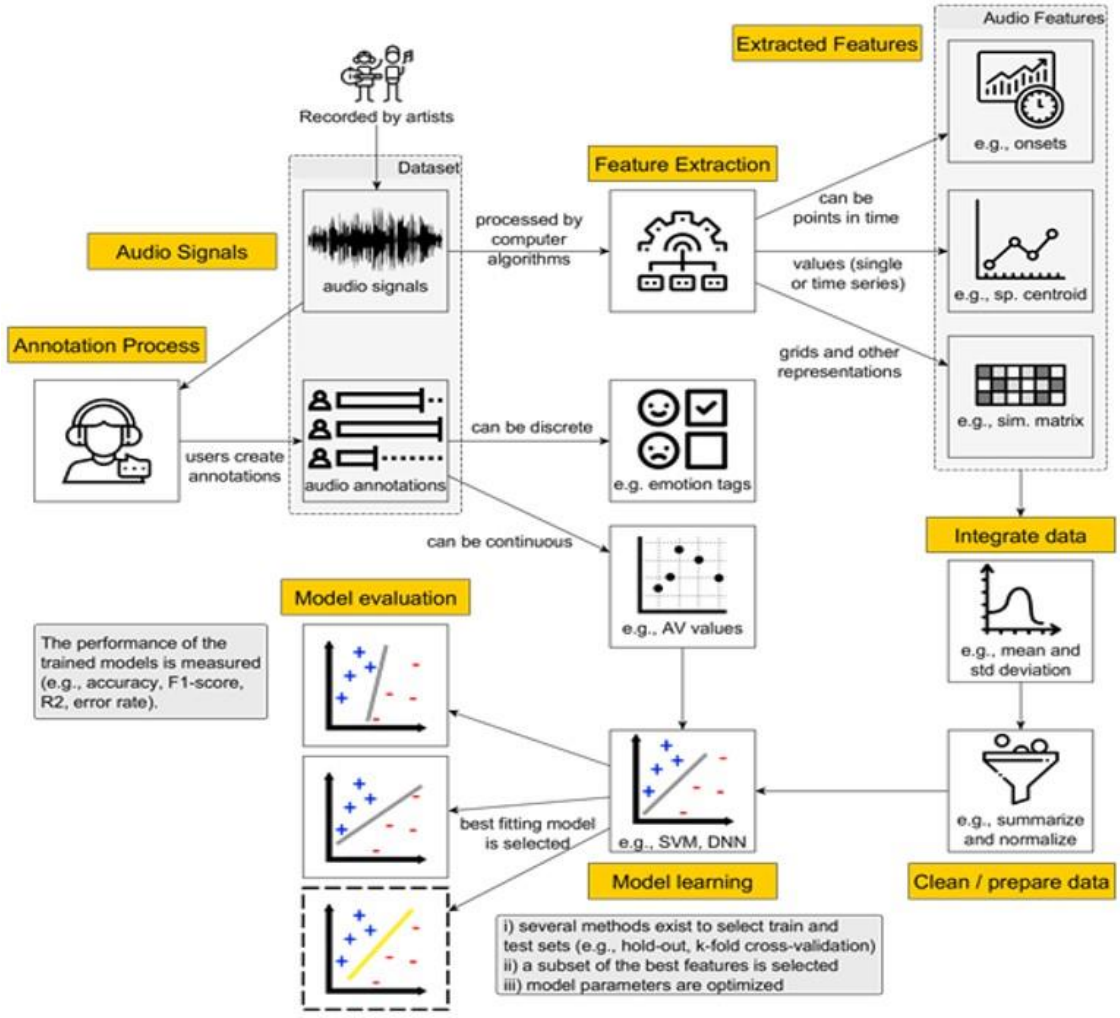


Figura 7 - Abordagem típica de aprendizagem computacional supervisionada aplicada em MER[22]

O processo inicia-se com a recolha de um conjunto de músicas e anotações que melhor descrevem o conteúdo emocional de cada música. A recolha das anotações é um processo complexo e sendo realizado por um grupo de pessoas voluntárias torna-se suscetível a erros que podem comprometer a qualidade dos dados. Outro dilema na obtenção de um *dataset* é o facto de habitualmente os ficheiros de áudio estarem sob leis de *copyright* muito restritas.

De seguida, os ficheiros de áudio são processados por algoritmos computacionais de modo a extrair algumas caraterísticas[23] (por exemplo, *zero-crossing rate*, que é o número de vezes

[22] Figura 3.3, retirada da página 102 de (R. E. S. Panda, 2019)
[23] descritores que melhor representam o conteúdo de uma música

que o sinal passa o eixo horizontal, ou seja, o número de vezes que o sinal muda de positivo para negativo ou vice-versa), integrando dados temporais (por exemplo a média). Podem ser extraídas todas as caraterísticas possíveis, mas isto não significa que devem ser todas utilizadas porque em (Hira & Gillies, 2015) é descrito que tendo demasiadas caraterísticas irrelevantes ao tema abordado, resulta num excesso de informação aumentando a complexidade do classificador.

A aprendizagem computacional é "a ciência (e arte) de programar computadores para que possam aprender com os dados" (Géron, 2019) e pode ser dividida em três partes como identificado anteriormente: treino, teste e classificação. A parte do treino é "a fase onde um algoritmo é treinado para criar a saída certa"[24], neste caso são identificados padrões entre as caraterísticas previamente extraídas dos ficheiros de áudio do *dataset* e as respetivas anotações emocionais. Habitualmente apenas um subconjunto de músicas é utilizado nesta parte, reservando as restantes para a validação do mesmo.

A fase de testes concretiza-se após a obtenção de um modelo treinado a fim de avaliar o desempenho do mesmo. O subconjunto de músicas do *dataset* que não foi utilizado previamente no treino é agora inserido no modelo para que o mesmo lhes atribua uma classificação emocional. As emoções obtidas pelo modelo nesta validação são comparadas com as anotações reais previamente identificadas no *dataset*, e através de algumas métricas é possível avaliar o desempenho do classificador.

A fase de classificação ocorre em ambiente real quando um utilizador pede a classificação de uma música. O sinal de áudio é descarregado, são extraídas as suas caraterísticas e posteriormente fornecidas ao modelo treinado a fim de obter uma classificação.

O processo descrito anteriormente era o processo utilizado nos primeiros sistemas automatizados de MER. Tipicamente era utilizada apenas a informação do sinal áudio, que até certo ponto já tinha demonstrado obter resultados satisfatórios (Laurier et al., 2008), mas mais tarde, os pesquisadores começaram a realizar alguns estudos utilizando a análise da

[24] https://medium.com/plumbersofdatascience/a-simple-explanation-of-the-machine-learning-workflow-c5d43d9f5b1c

lírica, juntamente com a análise do seu áudio, resultando em sistemas bi-modais com melhor precisão (Malheiro et al., 2016). Apesar de permitirem obter melhores resultados, estas abordagens são particularmente difíceis porque a extração de caraterísticas e o esquema para etiquetar as emoções não é trivial, ainda mais considerando a complexidade envolvida nas ambiguidades do texto (Kim et al., 2010). Ainda assim, alguns sistemas começaram também a extrair caraterísticas da lírica, como por exemplo, caraterísticas de semântica.

Outros estudos, como por exemplo (Xu et al., 2014), começaram a implementar a separação de fontes num áudio, obtendo resultados claramente superiores. A separação de fontes consiste em recuperar cada sinal original (voz ou componente de guitarra por exemplo) a partir do sinal combinado. É possível que através de determinadas caraterísticas, a componente vocal seja suficiente para distinguir se uma música é 'calma' ou 'triste', mas esta eficácia perde-se quando as componentes são misturadas (R. Panda et al., 2020b). As emoções nas músicas são habitualmente expressas em pequenos excertos e, por isso, é uma prática comum segmentar cada áudio em pequenos excertos com tamanho fixo (Xu et al., 2014), onde depois será executada a extração de caraterísticas, como ilustrado na Figura 8. Mas a literatura não é clara relativamente à duração dos excertos, no caso de *datasets pop*, por exemplo, costumam ser utilizados 25-30s mas há estudos que apontam para 8-16secs (Xinyu Yang et al., 2018).

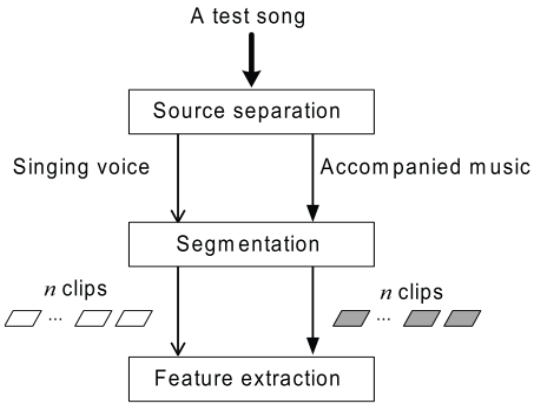


Figura 8 - Possível diagrama para separação de fontes num sistema de MER[25]

[25] Figura 1, retirada da página 1 de (Xu et al., 2014)

2.5. *Dataset*

Um *dataset* é um conjunto de dados estruturados. Estes dados não têm de ser necessariamente texto ou valores, podem ser coleções de imagens ou vídeos por exemplo. É habitual que os dados sejam estruturados sob forma de tabelas, onde cada coluna representa uma variável específica e cada linha representa cada membro do conjunto de dados.

Para o propósito deste trabalho, seguindo um pouco os estudos que se vêm a realizar na área, são extraídas caraterísticas do sinal de áudio e da lírica, e por isso optou-se por utilizar dois *datasets* distintos, um para cada componente. Um dos conjuntos de dados é constituído por ficheiros de áudio e o outro por ficheiros de texto com líricas de músicas. Ambos os conjuntos estão devidamente etiquetados com as emoções de cada música. Encontra-se descrito de seguida qual o conjunto utilizado para ambas as componentes.

2.5.1. Áudio

Existem alguns *datasets* públicos que contêm áudios e poderiam ser utilizados, mas o conjunto escolhido[26] foi criado no âmbito do projeto de investigação MOODetector e fruto de um trabalho de doutoramento (R. E. S. Panda, 2019), por forma a validar o trabalho proposto de desenvolvimento de novas características musicais. Este conjunto de dados é constituído por 900 ficheiros de áudio, em que cada excerto tem cerca de 30 segundos de duração e as respetivas emoções seguem a taxonomia categórica, utilizando quatro classes para representar cada um dos quatro quadrantes do modelo *Russell's Circumplex Model of Emotion* (Figura 6).

O processo de criação do conjunto de dados foi bastante controlado e está descrito na secção 4.1 de (R. E. S. Panda, 2019), consistindo inicialmente em descarregar várias músicas, etiquetadas com diferentes emoções (289 possíveis) por especialistas, utilizando a API (Application Programming Interface) da plataforma AllMusic[27]. Posteriormente foi realizado um processo de validação cego, em que voluntários ouviram os excertos e

[26] *Link* onde pode ser descarregado o *dataset*: http://mir.dei.uc.pt/downloads.html
[27] https://www.allmusic.com/

etiquetaram a emoção que identificavam nos mesmos em quadrantes - emoção que entendiam estar a ser transmitida pela música, não a emoção sentida, pois esta é mais pessoal e subjetiva. Por fim, as anotações obtidas foram cruzadas com as originais vindas do AllMusic e mantidas aquelas onde houve concordância. Todas as músicas em que não era possível identificar a emoção de forma clara ou em que a qualidade do som não era suficientemente boa foram descartadas (ex., versões ao vivo com aplausos ou outros sons), resultando então em 900 excertos de 30 segundos, 225 excertos para cada classe (quadrante).

2.5.2. Lírica

O *dataset* público escolhido[28] para a análise da lírica foi proposto em (Malheiro et al., 2016) para validar as novas caraterística propostas no âmbito de identificar emoções na lírica. Este *dataset* é constituído por dois conjuntos de músicas, um com 180 e o outro com 771, totalizando 951 líricas. Assim como o *dataset* a utilizar para a análise do áudio, também aqui as emoções seguem a taxonomia categórica, utilizando quatro classes para representar os quadrantes do modelo *Russell's Circumplex Model of Emotion* (Figura 6). As líricas das músicas escolhidas foram obtidas a partir de três *sites* de informações de lírica, sendo depois pré-processadas por forma a melhorar a qualidade dos dados através de algumas tarefas como por exemplo a correção de erros ortográficos. A identificação das emoções foi realizada num estado inicial pela própria equipa, onde uma emoção foi pré-anotada se pelo menos um membro da equipa estivesse de acordo com a etiqueta atribuída na plataforma Last.FM[29] à respetiva música. Posteriormente a identificação das emoções foi realizada por trinta e nove pessoas com diferentes formações académicas e géneros sexuais. As músicas estão bem balanceadas pelos quatro quadrantes.

2.6. Extração de caraterísticas

A extração de caraterísticas de uma música é um processo essencial na abordagem típica de MER porque é através destes descritores que será possível fazer a classificação. As

[28] *Link* onde pode ser descarregado o *dataset*: http://mir.dei.uc.pt/downloads.html
[29] https://www.last.fm/

caraterísticas permitem fazer a distinção entre objetos, fornecendo as primitivas essenciais para identificar inequivocamente um objeto (R. E. S. Panda, 2019). O sistema de extração é complexo, devido à grande quantidade de informação presente num sinal, sendo bastante difícil conseguir identificar e obter toda a informação útil. Mas ainda assim é a base para diferentes áreas de investigação, como o reconhecimento de emoções em música (R. E. S. Panda, 2019).

Nas últimas décadas foram desenvolvidas várias *frameworks* que possuem algoritmos computacionais para obter informações do sinal de áudio. Algumas das *frameworks* mais utilizadas são brevemente descritas e abreviadas de seguida. Informações mais detalhadas estão presentes em (Moffat et al., 2015).

Em relação a *frameworks* para extração de lírica, sendo que na prova de conceito anterior (António, 2019) não foi considerada a componente de lírica e por isso encontra-se no estágio inicial para deste trabalho, foi apenas avaliada uma única *framework*.

2.6.1. *Frameworks* de áudio

Essentia

Essentia é uma biblioteca de código aberto escrita na linguagem de programação C++. É multiplataforma e pode ser utilizada num ambiente de Python, JavaScript e através da linha de comandos. Implementa uma ampla coleção de algoritmos reutilizáveis, entre eles permite a caraterização de dados estatísticos e obter vários descritores musicais espetrais, temporais, de tom e alto-nível. Todos os algoritmos visam analisar, descrever e sintetizar o sinal de um áudio, focando-se em otimizar a velocidade computacional e a memória utilizada. Tem ainda outros algoritmos para complementar a extração de caraterísticas musicais, como por exemplo o algoritmo YamlOutput que permite o armazenamento das caraterísticas musicais em ficheiros com o formato JSON (JavaScript Object Notation) ou YAML (YAML Ain't Markup Language) e posteriormente serem utilizadas se necessário.[30]

[30] http://essentia.upf.edu/documentation.html

Librosa

Librosa é uma biblioteca em Python para processamento de sinais de áudio e "fornece a implementação de várias funções comuns utilizadas na área de MIR" (Mcfee et al., 2015). É multiplataforma e os seus principais objetivos são: ter uma curva de aprendizagem reduzida para os indivíduos que têm conhecimentos em MATLAB, garantir um padrão nas interfaces, nomes de variáveis e nos parâmetros por defeito das funções de processamento, manter a compatibilidade com versões anteriores sempre que possível, permitir que investigadores façam melhorias em componentes específicos das funções, criando as suas próprias funções se assim optarem, porque como a área de MIR está em constante evolução tende a necessitar de sofrer alterações, ter o código legível através de documentação e baterias de testes exaustivas (Mcfee et al., 2015). Está disponível em (Mcfee et al., 2015) uma explicação detalhada sobre a biblioteca, como por exemplo como está organizada ou otimização de parâmetros para as várias funções.

MIR Toolbox

MIR Toolbox oferece um conjunto de funções escritas em MATLAB dedicadas à extração de caraterísticas musicais de baixo e alto nível em sinais de áudio. O seu objetivo é oferecer uma visão geral de abordagens computacionais na área de MIR. O projeto baseia-se numa estrutura modular: os vários algoritmos são decompostos em estágios e formalizados utilizando um conjunto reduzido de mecanismos elementares. Estes mecanismos elementares integram variantes propostas por abordagens alternativas de investigadores que podem ser selecionados e parametrizadas pelos utilizadores.[31]

A escolha de um *design* orientado a objetos permite ter uma grande flexibilidade em relação à sintaxe, mas o facto da *toolbox* ser escrita em MATLAB torna a *framework* bastante pesada em termos computacionais.

[31] https://www.jyu.fi/hytk/fi/laitokset/mutku/en/research/materials/mirtoolbox

YAAFE

Yaafe (Yet Another Audio Feature Extractor) é um programa de linha de comandos para executar extração de caraterísticas musicais. Foi projetado para corresponder aos seguintes requisitos: eficiência computacional com uma exploração apropriada de cálculos redundantes de caraterísticas musicais, simplicidade na utilização, capacidade para processar ficheiros de áudio bastante longos e simplicidade e eficiência no armazenamento (Mathieu et al., 2010). É ainda disponibilizada uma API escrita em C++ que pode ser utilizada num ambiente de Python ou MATLAB.[32] Esta disponível em (Mathieu et al., 2010) uma explicação detalhada.

Considerações Finais

Existem várias *frameworks* disponíveis para executar a extração de caraterísticas musicais.[33] As caraterísticas podem ser por exemplo de baixo nível, que são caraterísticas mais próximas do sinal de áudio e do seu espectro, pouco percetíveis para o ser humano, ou podem ser por exemplo caraterísticas de alto nível, estando mais perto daquilo que o ser humano retira de uma música.

As *frameworks* de extração de características musicais têm sido muito utilizadas a longo dos últimos anos em vários estudos. Com base nas informações anteriormente descritas, na Tabela 1 que ilustra as principais características das *frameworks* estudadas e na Figura 9 que ilustra o tempo de computação necessário para executar a extração de caraterísticas de cada *framework* num determinado ambiente, optou-se por escolher a biblioteca Essentia. A escolha recaiu principalmente sobre os motivos de: a biblioteca fornecer um variado conjunto de funcionalidades para extração de caraterísticas, pelo facto de ser desenvolvida em C++ que permite otimizar os recursos computacionais necessários em comparação com MATLAB por exemplo, o tempo necessário para executar a extração das caraterísticas e a variedade de linguagens disponibilizada para ter acesso à biblioteca.

[32] http://yaafe.sourceforge.net/
[33] http://www.ismir.net/resources/software-tools/

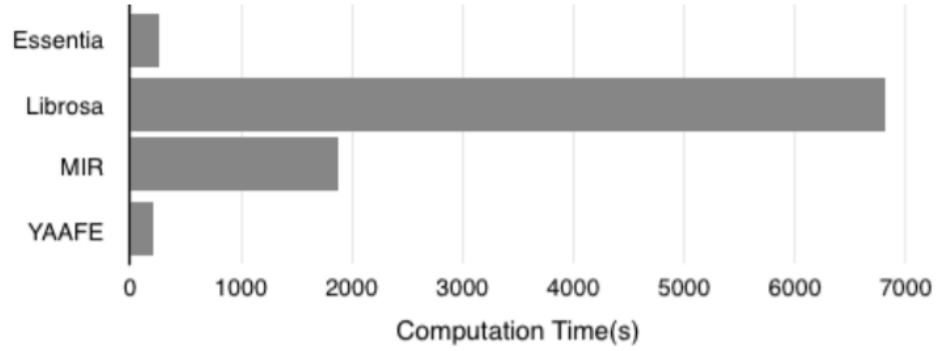


Figura 9 – Gráfico do tempo de computação necessário para a extração de caraterísticas de cada *framework*[34]

	Essentia	**Librosa**	**MIR Toolbox**	**YAAFE**
Caraterísticas de alto nível	Sim	Sim	Sim	Sim
Caraterísticas de baixo nível	Sim	Sim	Sim	Sim
Filtros	Sim	Não	Sim	Não
API	C/C++ Python MATLAB	Python	MATLAB	C/C++ Python MATLAB
Output	YAML JSON	CSV	TSV ARFF	CSV HDF5

Tabela 1 - Comparação das diferentes ferramentas de extração de características[35]

2.6.2. Lírica

As emoções de uma música são expressas através de caraterísticas musicais, mas uma parte importante também parece ser transmitida através da lírica (Laurier et al., 2008). Algumas destas informações transmitidas podem até ser exclusivas da lírica, ou seja, as caraterísticas

[34] Figura baseada na figura 2, retirada da página DAFX-5 de (Moffat et al., 2015)
[35] Tabela baseada na tabela 1, retirada da página DAFX-4 de (Moffat et al., 2015)

do áudio podem não refletir esta informação, perdendo-se assim quando os sistemas apenas utilizam o áudio para identificar emoções.

Mas como este trabalho é mais focado no áudio e a componente da lírica nem sequer foi abordada na prova de conceito anterior (António, 2019), ainda está num estágio inicial relativamente a este trabalho e, por esse motivo, foi apenas avaliada uma *framework* para extração de caraterísticas de lírica.

A *framework* em questão é um projeto[36] que foi desenvolvido para o Centro de Informática e Sistemas da Universidade de Coimbra (CISUC) por dois alunos, Hugo Redinho e Carolina Gonçalves. A escolha por esta *framework* deve-se ao facto dos estudantes estarem a realizar o projeto sobre a orientação do Professor Doutor Renato Panda, que também orienta este mesmo trabalho e, por isso, é mais fácil entrar em concordância e articular as ideias com eles consoante as necessidades.

O projeto consiste num extrator de características de letras de músicas (*music lyrics*) baseando-se nos seguintes tipos de caraterísticas: semântica, baseadas na estrutura e no conteúdo. Cada um destes tipos descritos pretende extrair diversas caraterísticas da área especifica. Está desenvolvido em JAVA, podendo ser utilizado o seu ambiente gráfico para proceder à extração ou então usar o executável através da linha de comandos. É possível extrair as várias caraterísticas individualmente ou todas de uma única vez.

2.7. Algoritmos de aprendizagem computacional

A componente do classificador é um dos últimos processos a ser realizado. Neste momento admite-se que já foram extraídas as caraterísticas do sinal de áudio e, com base nestas caraterísticas, é utilizada a aprendizagem computacional por forma a obter uma possível emoção que caraterize o áudio. Sistemas com aprendizagem computacional podem seguir quatro categorias diferentes: aprendizagem supervisionada (*supervised learning*), não supervisionada (*unsupervised learning*), semi-supervisionada (*semisupervised learning*) e

[36] https://github.com/hugoredinho/FeatureExtractionSystem

aprendizagem por reforço (*reinforcement learning*) (Géron, 2019). Este trabalho segue a abordagem de classificação proposta em (R. Panda et al., 2020b), que já tinha sido implementada na primeira versão deste projeto (António, 2019), e tem por base a categoria de aprendizagem supervisionada que consiste em incluir nos dados de treino as soluções pretendidas para cada registo, a sua emoção, para fornecer ao algoritmo de classificação como mostra a Tabela 2. Consultar (Géron, 2019) para mais informações sobres as várias categorias de aprendizagem computacional.

Registo	**Caraterística 1**	**Caraterística 2**	**…**	**Caraterística N**	**Classe**
1	xx	xx		xx	Feliz
2	xx	xx		xx	Triste
3	xx	xx		xx	Tensa
…	…	…		…	…

Tabela 2 - Instâncias com soluções pretendidas[37]

Existem vários algoritmos de aprendizagem supervisionada e a escolha de qual o algoritmo a utilizar é um passo crucial para o sistema. Assim, são analisados de seguida alguns dos mais importantes algoritmos para aprendizagem computacional supervisionada sendo eles *k-Nearest Neighbors*, *Support Vector Machines* (SVM) e *Decision Trees* (Árvores de Decisão).

K-Nearest Neighbors

K-Vizinhos mais próximos (*K-Nearest Neighbors*, KNN) é um algoritmo para aprendizagem computacional supervisionada que se baseia na premissa de que "as instâncias num conjunto de dados normalmente existem bastante próximas a outras instâncias que têm propriedades idênticas" (Kotsiantis, 2007). As instâncias podem ser consideradas pontos num plano com n-dimensões, onde os eixos do referencial representam as caraterísticas extraídas.

[37] Baseada na tabela 1, da página 249, de (Kotsiantis, 2007)

Este algoritmo é bastante poderoso, mas existem algumas questões mais notórias a ter em conta: 1) os requisitos de armazenamento são enormes; 2) é sensível à escolha da função de similaridade que é utilizada para comparar instâncias; 3) carece de uma maneira de escolher o K, exceto através de métodos de validação cruzada ou outras técnicas de semelhança computacionalmente árdua, e o desempenho do algoritmo é afetado dependendo do método que é utilizado. Estas questões identificadas são abordadas em detalhe em (Kotsiantis, 2007).

Ao identificar as instâncias de treino com etiquetas, é possível prever a etiqueta de uma nova instância avaliando a sua colocação no plano do referencial. A nova etiqueta é determinada segundo a etiqueta que está em maioria nos seus K vizinhos mais próximos. A Figura 10 permite visualizar este processo, representando um referencial de duas dimensões com diferentes dados associados. O ponto a vermelho é um dado não classificado e os pontos a amarelo e roxo são classificados com Classe A e Classe B respetivamente. Admitindo que o K teria o valor 3 e calculando a distância entre o novo dado e os vizinhos mais próximos, são identificados os três pontos que estão mais próximos e analisadas as suas etiquetas por forma a identificar a etiqueta mais recorrente, que neste caso seria Classe B e por isso seria atribuída ao novo dado.

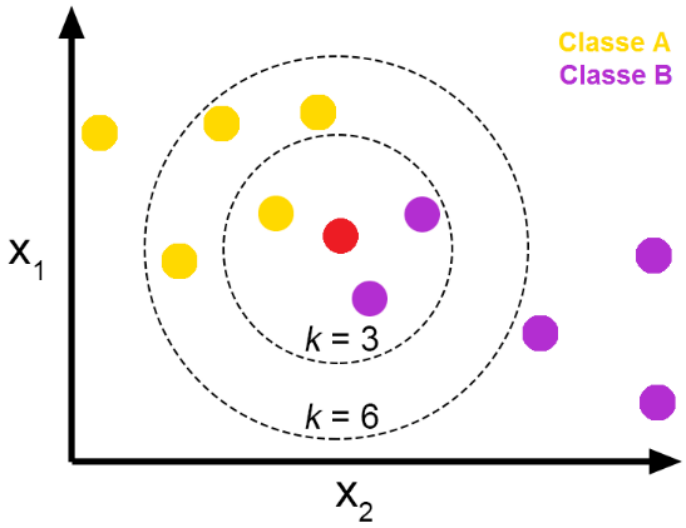


Figura 10 - K-Nearest Neighbors[38]

[38] Retirada de https://medium.com/brasil-ai/knn-k-nearest-neighbors-1-e140c82e9c4e

Support Vector Machines

Máquinas de vetor de suporte (Support Vector Machines, SVM) são dos modelos mais populares de aprendizagem computacional e particularmente muito adequados para a classificação de pequenos ou médios conjuntos de dados (Géron, 2019). O objetivo das SVMs é obter um plano que separe o melhor possível os dados das diferentes classes, sendo que a *kernel* define as propriedades desse plano, aplicando transformações nos dados, como mostra a Figura 11. As quatro funções *Kernel* existentes são *linear*, *poly* (*polynomial)*, *rbf* (*Radial Basis Function*) e sigmoid. Cada uma destas funções é combinada com vários outros parâmetros específicos de cada *kernel*, onde o *C* corresponde a *cost* (custo) e regulariza o termo de erro controlando a margem de decisão, um custo maior terá uma menor margem de decisão enquanto um custo menor tem uma margem maior. O parâmetro gama decide a curvatura desejada nos limites de decisão, onde uma gama maior significa maior curvatura. O *kernel* polinomial tem um parâmetro adicional que é o *degree* e este detalha o grau da função polinomial a utilizar.

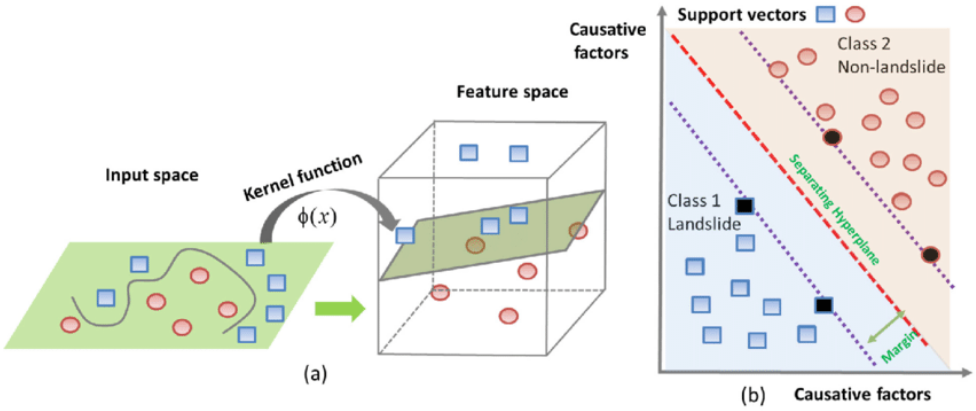


Figura 11 - Ilustração do princípio de SVMs[39]

SVM é capaz de realizar classificação linear e não-linear. Classificação linear consiste em separar as classes facilmente utilizando uma ou mais linhas (vetores), sendo assim linearmente separáveis como mostra a Figura 12. Nesta figura existem duas classes e três

[39] Figura retirada de https://www.researchgate.net/publication/331801455_Evaluating_GIS-Based_Multiple_Statistical_Models_and_Data_Mining_for_Earthquake_and_Rainfall-Induced_Landslide_Susceptibility_Using_the_LiDAR_DEM

possíveis vetores para as separar. Obviamente o vetor a tracejado verde é mau porque não separa as classes corretamente. Qualquer um dos outros dois vetores seriam uma possibilidade válida para utilizar, mas é provável que o desempenho de ambos não seria ideal para novas instâncias porque as instâncias representadas encontram-se demasiado perto do vetor, ou seja, do limite de decisão. Já na Figura 13 encontra-se representada uma linha sólida a preto que seria a ideal, porque para além de separar as classes, também se encontra o mais longe possível das instâncias, oferecendo uma margem maior para decisão.

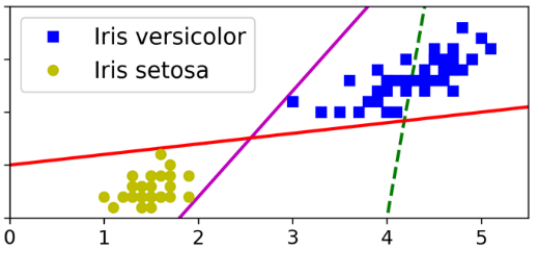


Figura 12 - Três possíveis classificadores lineares[40]

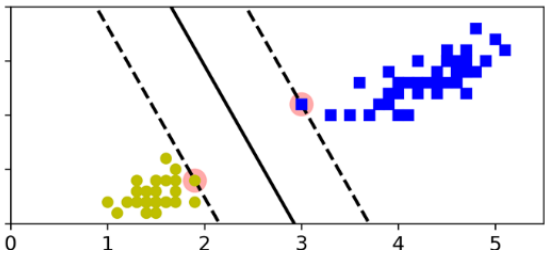


Figura 13 - Limite de decisão de um classificador SVM[41]

Nem todos os conjuntos de dados podem ser separados linearmente e esta situação designa-se de classificação não-linear. Quando esta situação ocorre, uma das abordagens a utilizar é adicionar novas caraterísticas para se conseguir separar as classes, sendo que por vezes é possível obter um novo conjunto de dados linearmente separável. Existem vários tipos de caraterísticas que se podem adicionar, como exemplo caraterísticas polinomiais. Estas informações estão mais detalhadas em (Géron, 2019).

Decision Trees

Árvores de decisão (*decision trees*) são "árvores que classificam as instâncias dispondo-as através do valor das caraterísticas" (Kotsiantis, 2007), ou seja, é um algoritmo de aprendizagem computacional que se baseia na lógica de tomadas de decisão.

[40] Figura 5-1, retirada da página 219, de (Géron, 2019)
[41] Figura 5-1, retirada da página 219, de (Géron, 2019)

Uma árvore, como a exemplar da Figura 14, é constituída por nós (*nodes*) e ramos, sendo que cada nó representa uma caraterística e cada ramo representa um possível valor que o nó pode assumir. O nó do topo da árvore é designado por nó raiz (*root*) e é por este que o processo se inicia, seguindo um fluxo vertical de cima para baixo, tomando as decisões consoante o valor dos ramos. Tendo como exemplo a Figura 14, verifica-se que inicialmente existem três caminhos (ramos) possíveis de seguir e a decisão depende dos valores dos ramos, podendo o processo acabar de imediato, para o caso de se optar pelos ramos *b1* e *c1*. Se o ramo escolhido for *a1*, a decisão continua para o nó *at2*, repetindo-se o processo de decisão.

Estes algoritmos são poderosos e versáteis, ajustando-se bastante bem a conjuntos de dados complexos, permitindo a realização de tarefas de classificação e regressão (Géron, 2019).

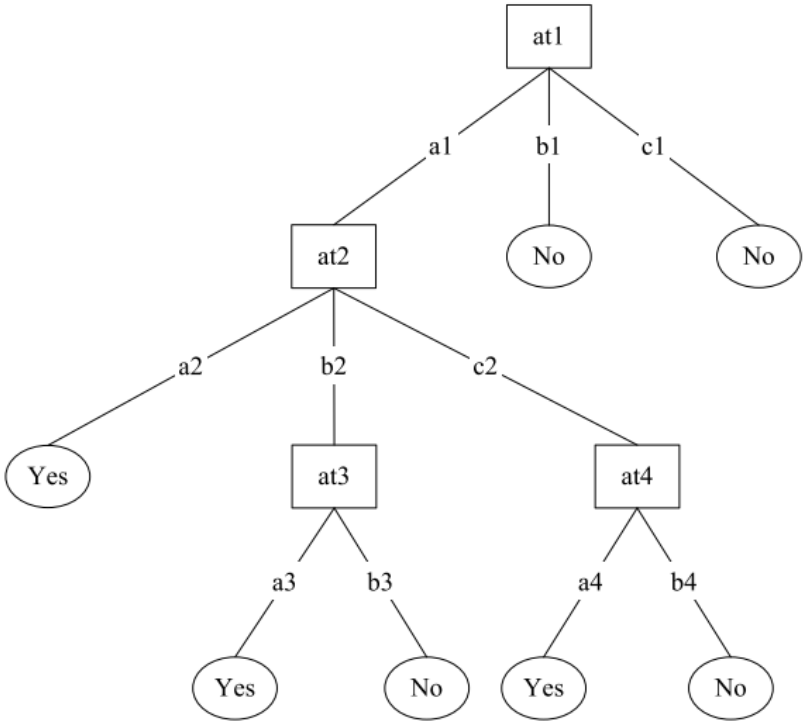


Figura 14 - Árvore de decisão[42]

[42] Figura 2, retirada da página 251, de (Kotsiantis, 2007)

Considerações Finais

São vários os algoritmos de aprendizagem computacional disponíveis para uso, sendo os três algoritmos descritos acima apenas alguns dos mais conhecidos. Para além de algoritmos de aprendizagem computacional tradicionais, também foi avaliada a possibilidade de utilizar algoritmos de aprendizagem profunda (*Deep Learning*), mas esta técnica necessita de conjuntos com imensos dados anotados e grande capacidade de computação (com GPUs), o que ainda não se verifica atualmente.

Os três algoritmos considerados são possíveis de utilizar, mas a escolha recaiu sobre SVM. Como suporte à decisão, para além da informação recolhida e descrita já anteriormente, encontram-se especificados alguns detalhes sobre os algoritmos na Tabela 3, onde **** representa melhor desempenho e * representa pior desempenho. Os resultados desta tabela servem de guia, podendo qualquer um dos algoritmos funcionar melhor em problemas específicos. A precisão obviamente tem bastante impacto na escolha de um algoritmo e para SVM é considerada o melhor valor entre os algoritmos avaliados. SVM demonstra também uma boa avaliação a nível de tolerância aos diferentes aspetos representados.

	Árvores de Decisão	**kNN**	**SVM**
Precisão	**	**	****
Velocidade de aprendizagem	***	****	*
Velocidade de classificação	****	*	****
Tolerância a valores em falta	***	*	**
Tolerância a caraterísticas irrelevantes	***	**	****
Tolerância a caraterísticas redundantes	**	**	***
Tolerância a ruído	**	*	**

Tabela 3 - Comparação entre algoritmos de aprendizagem automática para problemas genéricos segundo (Kotsiantis, 2007)

O facto deste trabalho se inspirar bastante nas decisões tomadas em (R. Panda et al., 2020b) também impactou bastante na decisão, tendo em conta a opção tida no respetivo. Ainda mais porque este já era o algoritmo em utilização na prova de conceito anterior (António, 2019).

2.7.1. *Frameworks* para implementação de SVM

Como descrito na secção 2.7, a aprendizagem computacional baseia-se em algoritmos. Estes algoritmos apoiam-se em matemática e estatística sendo bastante complexos e difíceis de compreender para um utilizador com poucos conhecimentos na área. Por esse motivo são desenvolvidas e disponibilizadas *frameworks* que permitem criar modelos de aprendizagem computacional sem um conhecimento tão profundo sobre o tema[43]. De seguida são descritas algumas das *frameworks* mais conhecidas e utilizadas atualmente, sejam elas bibliotecas ou ferramentas, e qual a *framework* escolhida para implementar SVM considerando diversos fatores.

TensorFlow

TensorFlow (TF) é uma biblioteca grátis de código aberto destinada a qualquer utilizador independente do seu conhecimento em aprendizagem computacional e centra-se na computação numérica distribuída. Permite realizar, utilizando múltiplos CPUs e GPUs, diversas tarefas como regressão e classificação, incluindo SVM, mas o âmbito principal é no treino de redes neuronais e quando associada a aprendizagem computacional é essencialmente focada em grandes conjuntos de dados.

É multi-plataforma, podendo ser utilizado especificamente em ambientes Linux, macOS, Windows, e ainda em plataformas de computação móvel incluindo Android e iOS. Disponibiliza APIs estáveis em Python e C, e tem vários pacotes (*packages*) disponíveis em várias outras linguagens de programação onde a compatibilidade não é garantida.[44]

[43] https://www.bmc.com/blogs/machine-learning-ai-frameworks/
[44] https://en.wikipedia.org/wiki/TensorFlow

Scikit-Learn

Scikit-Learn é uma biblioteca grátis para Python que permite executar várias tarefas úteis em aprendizagem computacional, entre elas regressão e classificação, incluindo os algoritmos descritos na secção 2.7. Foi projetada com a intenção de desempenhar em conjunto com as bibliotecas numéricas e científicas NumPy e SciPy. Fornece também diversas ferramentas simples, eficazes e bastante vantajosas para a análise de modelos, como as matrizes de confusão para saber quão bem o modelo desempenhou.

A biblioteca é desenvolvida em Python porque é uma linguagem de programação de alto nível, tornando a componente de aprendizagem computacional mais abstrata para o utilizador não especialista e ainda para maximizar a eficiência computacional. Esta é uma biblioteca que é habitualmente utilizada com pequenos conjuntos de dados. Para mais informações consultar (Pedregosa et al., 2011).

Considerações Finais

Ambas as *frameworks* estudadas são bastante poderosas e justificam porque são das mais famosas. Entre as duas, optou-se por utilizar Scikit-learn. Os principais motivos que sustentam esta decisão devem-se ao facto de que provavelmente a biblioteca TensorFlow é um exagero para o estado atual e para o que se pretende deste sistema. Esta biblioteca, sendo de mais baixo nível, requereria conhecimentos mais aprofundados na área não só para interagir e utilizar a mesma, mas também para montar modelos de aprendizagem computacional. Para além disso é essencialmente associada a grandes conjuntos de dados, que atualmente não é caso.

Scikit-learn sendo de mais alto nível, disponibiliza diversos algoritmos de aprendizagem computacional que são possíveis de utilizar e ainda fornece diversas ferramentas ótimas para análise de modelos. Num estágio mais avançado do trabalho, será possível transitar a utilização TensorFlow se for devidamente justificável.

2.8. Separação de fontes (*source separation*)

A separação de fontes consiste em recuperar os elementos sonoros individuais (ex.: voz, guitarra ou acompanhamento) a partir do sinal combinado onde estão misturados. Foram estudadas três bibliotecas, descritas de seguida, para fazer este processo: NUSSL, Demucs e Spleeter.

NUSSL

NUSSL (Northwestern University Source Separation Library) é uma biblioteca de código aberto, implementada em Python e orientada à separação de elementos num sinal de áudio. Esta biblioteca contém algoritmos comuns para separação de fontes, vários recursos para modelos em *deep learning* e uma estrutura de fácil utilização para prototipagem e adição de novos algoritmos.

O objetivo da biblioteca é "permitir o uso de algoritmos populares para separação de fontes, enquanto permite que o utilizador tenha controlo sobre parâmetros de baixo nível"[45].

Com base na documentação e numa demonstração[46] é possível verificar que o processo para fazer a separação de fontes com esta biblioteca é relativamente simples e conta com bastantes algoritmos para serem utilizados. O resultado da separação consiste em dois áudios, o primeiro e segundo plano do áudio. O primeiro plano é composto habitualmente por sons com caraterísticas espetrais que variam rapidamente enquanto o segundo plano (ou plano de fundo) consiste em caraterísticas espetrais que variam lentamente.

Em suma, verifica-se que o ficheiro do primeiro plano é constituído maioritariamente pela componente vocal e o segundo plano é constituído fundamentalmente pelo acompanhamento musical, mas a qualidade de ambos os planos é baixa, existindo bastante ruído. Para informações mais detalhadas consultar (Manilow et al., 2018).

[45] https://nussl.github.io/docs/
[46] https://interactiveaudiolab.github.io/demos/nussl.html

Demucs

Demucs baseia-se na arquitetura convencional U-NET (rede neuronal) inspirada em Wave-U-Net[47] e SING[48]. Através de modelos pré-treinados, utilizando o *dataset* MusDB[49], permite separar os sinais individuais da bateria, baixo e voz, do resto do sinal misturado obtendo resultados de última geração, superando métodos anteriores baseados em *waveform* (forma da onda) ou espectrogramas, mas estes modelos são bastante pesados.

Pode ser executado em Linux, Windows e MAC OS X utilizando a distribuição Anaconda para executar o código. Permite a utilização tanto de CPU como de GPU, sendo que com a utilização de GPU para acelerar o procedimento são necessários pelo menos 8GB de RAM. O tempo de processamento utilizando o CPU é aproximadamente igual à duração da música. Para mais informações consultar (Défossez et al., 2019).

Spleeter

Spleeter é uma biblioteca para separação de fontes musicais, publicada pela companhia Deezer[50], e projetada a pensar na facilidade de utilização, no desempenho e velocidade da separação. A biblioteca é escrita em Python e possibilita treinar modelos para separação de fontes ou ajustar os modelos já existentes e ainda dividir os ficheiros de áudio em várias *stems* (fontes, elementos) utilizando os modelos pré-treinados:

- 2 *stems* – Separação em componente vocal e acompanhamento;
- 4 *stems* – Separação em componente vocal, baixo, bateria e outros;
- 5 *stems* – Separação em componente vocal, baixo, bateria, piano e outros.

A separação pode ser completamente executada utilizando CPU ou GPU e, como os modelos são baseados em redes neuronais, os cálculos são eficientemente paralelizados. A própria biblioteca, para maximizar o desempenho, identifica automaticamente se a máquina tem

[47] https://github.com/f90/Wave-U-Net
[48] https://github.com/facebookresearch/SING
[49] https://sigsep.github.io/datasets/musdb.html
[50] https://www.deezer.com/pt/company

GPU e, no caso de ter, faz a computação utilizando a biblioteca Tensorflow, senão utiliza a biblioteca Librosa porque é mais leve e rápida computacionalmente para não sobrecarregar o CPU[51]. Funciona em ficheiros de áudio com diferentes formatos (ex.: mp3 e wav) e o tempo de processamento é relativamente o mesmo para ambos. Mais detalhes em (Hennequin et al., 2020).

Foram realizados alguns testes localmente para avaliar a influência da memória RAM em relação ao tempo de processamento na separação em duas e quatros fontes, ilustrados respetivamente na Tabela 4 e Tabela 5. Os valores apresentados são o resultado de uma média de valores e todos os testes foram efetuados dentro de uma máquina virtual com 4 vCPUs/cores e a RAM respetiva indicada nas tabelas. A máquina virtual foi criada num computador portátil modelo HP OMEN 15-Ce018Np, processador Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz e RAM 16 GB. Não foram executados testes para separação em cinco fontes musicais porque as necessidades computacionais não permitiram a realização dos mesmos.

2 *Stems*			
	2GB	**4GB**	**6GB**
30 segundos	6,013s	6.15s	6.106s
1 minuto	8.3s	8.3s	8.25s
2 minutos	X	12.5s	12.42s
3 minutos	X	16.7s	16.45s
4 minutos	X	20.5s	20.2s
5 minutos	X	X	24.5s
6 minutos	X	X	28.3s

Tabela 4 - Tempo de processamento para 2 *stems*

4 *Stems*			
	2GB	**4GB**	**6GB**
30 segundos	11.33s	11.935	11,485
1 minuto	16.0s	15.7s	15.6s
2 minutos	X	23.1s	23.0s
3 minutos	X	30.7s	30.6s
4 minutos	X	X	38.3s
5 minutos	X	X	46.0s
6 minutos	X	X	X

Tabela 5 - Tempo de processamento para 4 *stems*

[51] https://github.com/deezer/spleeter/issues/348

Considerações Finais

A separação de fontes é algo que tem vindo a ser estudado na área de MER por forma a aumentar a precisão dos sistemas e, segundo os estudos, é realmente eficaz. Os componentes num áudio são habitualmente vários, pode ser a voz, guitarra, bateria, entre outros. Obviamente que na utilização de um modelo que faz a separação em duas fontes, vocal e acompanhamento, esta última componente é bem mais robusta que a vocal porque contém vários componentes. É normal que a separação dos elementos também não seja feita na perfeição porque é um processo complexo, ainda mais quando os áudios são apenas constituídos por instrumental por exemplo.

Existem diversas bibliotecas para obter estes elementos individualmente, e algumas destas foram estudadas anteriormente. Por forma a obter um pouco mais de informação prática sobre cada, foram ainda realizadas algumas separações nos áudios utilizando as diversas bibliotecas. Em relação à biblioteca NUSSL, sabe-se que apenas permite exercer a separação para duas componentes musicais, acrescentar que a qualidade da separação não é tão boa quando comparada com as outras duas bibliotecas e os requisitos computacionais necessários são elevados.

A biblioteca Demucs revelou ter a melhor qualidade de áudio entre as três bibliotecas, mas os modelos pré-treinados são bastante pesados, os recursos computacionais necessários são elevados, apenas permite exercer a separação de fontes em quatro fontes e o tempo de processamento é proporcional à duração de uma música, ou seja, se uma música tiver 3 minutos de duração, vai também demorar aproximadamente 3 minutos a fazer a separação utilizando CPU.

Spleeter tem uma qualidade de áudio boa, apesar de ser um pouco inferior ao Demucs, permite a separação em duas, quatro e cinco fontes, é relativamente rápido comparado com as outras bibliotecas e não são necessários tantos recursos computacionais, como ilustram as Tabela 4 e Tabela 5, e os modelos pré-treinados são relativamente leves.

Os resultados aos vários testes realizados poderiam sofrer alterações se fosse utilizada a GPU, pois todos os testes efetuados foram realizados usando apenas o CPU. Isto deve-se ao

facto de terem sido utilizadas máquinas virtuais para o ambiente de desenvolvimento e as máquinas virtuais não utilizam a GPU da máquina *host*, na verdade quase todo o *hardware* do *host* é virtualizado, assim como a placa gráfica[52]. Para ter acesso à placa gráfica na máquina virtual seria necessário fornecê-la por completo à máquina utilizando a ferramenta PCI Passthrough[53], mas não seria possível utilizar a mesma no *host* enquanto isto decorre, sendo este o principal motivo pelo qual se optou por não usar GPU.

Em suma, a escolhida recaiu sobre o Spleeter principalmente pela qualidade do áudio, que não é a melhor entre os três mas é bastante boa, por os recursos computacionais necessários serem os mais baixos, por oferecer variadas opções de separação, enquanto que os outros só permitiam separar em duas ou quatro fontes, sendo que se quiséssemos mudar o número de fontes era necessário escolher outra biblioteca e o tempo de processamento é bastante inferior comparando com as outras bibliotecas, e isto é algo que é essencial para o utilizador não ficar em espera demasiado tempo.

2.9. Obtenção de lírica

A lírica é uma componente musical que tem vindo a ser utilizada em diversos sistemas, originando soluções bi-modais, para aumentar a precisão do classificador. Isto não é algo surpreendente visto que assim como o áudio, também a lírica é uma componente musical e, dependendo do estilo de música, pode até ser uma dimensão mais relevante que o próprio áudio. São estudadas de seguida algumas tecnologias para obter a lírica de músicas.

Genius

Genius é uma empresa fundada em 2009 e inicialmente disponibilizava uma plataforma[54] para anotar letras musicais, mas apenas do género *rap*. Alguns anos depois, expandiram a sua missão para incluir anotações e interpretações de letras musicais, explicações e notícias,

[52] https://unix.stackexchange.com/questions/200165/how-to-make-virtual-box-use-video-memory-from-nvidia-graphics-card
[53] https://docs.oracle.com/en/virtualization/virtualbox/6.0/admin/pcipassthrough.html
[54] https://genius.com/

poesia e documentos. Atualmente é considerada uma das melhores bases de dados de conhecimento na internet, com as melhores letras musicais[55].

Disponibiliza uma API[56] que pode ser utilizada gratuitamente e contém diversos *endpoints* para utilizar, mas os que mais se identificam no âmbito deste trabalho são essencialmente criar, gerir e visualizar anotações de músicas, e obter informações sobre meta-dados das mesmas.

Lyrics.ovh

Lyrics.ovh é uma aplicação web[57] muito simples que permite procurar a letra de músicas de forma rápida. Disponibiliza uma API[58] com apenas um *endpoint* para a pesquisa de letras, tendo como parâmetros o nome do artista e o título da música.

A API pode ser utilizada gratuitamente, mas tem uma limitação, que não está documentada, relacionada ao número de pedidos, sendo que apenas é possível obter resposta aos pedidos que tenham entre eles um espaço de 120 segundos, ou seja, só é possível fazer pedidos com espaçamento de 2 minutos entre eles.

Google Programmable Search Engine + Custom Search JSON API

O serviço Programmable Search Engine[59] é disponibilizado pela Google e consiste em ter motores de pesquisa que permitem fazer consultas de acordo com as especificações do utilizador, onde é possível definir para pesquisar apenas em algumas páginas *web* individuais, aplicações *web* completas ou mesmo domínios. Este serviço permite fazer até 10 000 pesquisas diárias grátis. Assemelha-se bastante ao motor de pesquisa padrão da Google, mas este permite fazer as pesquisas apenas nas aplicações *web* pretendidas, como por exemplo *sites* de música, excluindo muita informação menos relevantes ao âmbito.

[55] https://genius.com/Genius-about-genius-annotated
[56] https://docs.genius.com/
[57] https://lyrics.ovh/
[58] https://lyricsovh.docs.apiary.io/#
[59] https://programmablesearchengine.google.com/about/

O Custom Search JSON API[60] também é um serviço da Google e permite aos desenvolvedores obter e mostrar, através de pedidos RESTful, os resultados das pesquisas do Programmable Search Engine, obtendo os resultados no formato JSON. Este serviço é muito mais restrito, permitindo apenas 100 pedidos diários grátis.

Imaginando a personalização do Programmable Search Engine com vários sites de música, seria possível depois fornecer esta pesquisa personalizada à Custom Search JSON API para obter informações em formato JSON, por exemplo a letra de músicas, dos sites especificados na pesquisa.

SpeechRecognition

SpeechRecognition é uma biblioteca em Python para reconhecimento da fala e suporta diversas APIs e *engines*, como por exemplo Google Cloud Speech API[61], Microsoft Azure Speech[62] ou IBM Speech to Text[63].

Esta biblioteca oferece várias funcionalidades para o reconhecimento da fala, entre elas, a possibilidade de reconhecer a fala num ficheiro de áudio e transcrever a mesma para um ficheiro de texto.

Apesar da precisão destes sistemas que utilizam a técnica de Speech Recognition ter aumentado, ainda está longe da perfeição e, por isso, existe sempre a possibilidade de existirem erros no reconhecimento. Para além disso, a voz cantada é diferente do discurso normal, pelo que transcrição desta acrescenta maior dificuldade, sendo um problema de investigação em aberto.

Para além das limitações descritas acima, as várias APIs ou *engines* suportadas também têm limitações nos seus planos grátis, por exemplo, a duração limite para utilizar o serviço mensalmente.

---

[60] https://developers.google.com/custom-search/v1/overview
[61] https://cloud.google.com/speech-to-text
[62] https://azure.microsoft.com/en-us/services/cognitive-services/speech-services/
[63] https://www.ibm.com/cloud/watson-speech-to-text

Considerações Finais

As abordagens baseadas em lírica não são muitas, principalmente devido à sua complexidade, mas a sua utilização tem vindo a crescer devido ao peso emocional que a lírica pode proporcionar num áudio. Existem diversas tecnologias para obter a lírica, principalmente APIs que disponibilizam *endpoints* fazendo uso de bases de dados com uma enorme quantidade de informação relacionada à música, incluindo líricas. Algumas das possíveis tecnologias foram descritas anteriormente.

A tecnologia de reconhecimento da fala (Speech Recognition) oferece a possibilidade de reconhecer a fala num ficheiro de áudio e transcrever a mesma para um ficheiro de texto. Apesar de permitir a execução do que era proposto, este tipo de mecanismo ainda está longe da perfeição sendo suscetível a diversos erros. Para além disto, também é bastante limitada nos seus planos grátis e por isso não foi selecionada.

O serviço disponibilizado pela Google é sem dúvida bastante poderoso e serviria perfeitamente para superar as necessidades. Todavia este serviço é igualmente bastante limitado no seu plano grátis e, portanto, optou-se por não utilizar o mesmo.

As APIs da Genius e da Lyrics.ovh permitiam as duas obter a lírica, mas a Genius é bastante mais robusta e aparentemente não tem limites nos pedidos e, por estes motivos, foi a escolhida para obter a lírica.

Capítulo 3
Classificação de emoção em música

O sistema tem por objetivo obter a classificação de um áudio e da lírica através das caraterísticas extraídas. Pretende-se ter um classificador para a lírica e quatro classificadores para o áudio que servem para obter a emoção da componente vocal, do acompanhamento, do áudio original e das componentes de acompanhamento, original e vocal juntas. Este capítulo descreve o processo de investigação em reconhecimento de emoções em música. Para tal, partimos do conhecimento adquirido no estado de arte e com base neste efetuamos as etapas típicas de um sistema de *machine learning*, como a preparação do conjunto de dados, extração de características e execução de testes, tendo como objetivo final a criação dos modelos treinados a incluir no sistema desenvolvido. Com base na análise da secção 2.7.1, está a ser utilizada a *framework* scikit-learn[64] para todas as tarefas relacionadas com a classificação, por esse motivo os vários *scripts* necessários foram desenvolvidos em Python.

3.1. Preparação dos conjuntos de dados

Para obter classificações são necessários modelos previamente treinados. Se em ambiente real são extraídas as caraterísticas dos áudios fornecidos pelo utilizador, segundo um formato, duração e propriedades específicas, então os modelos de classificação previamente treinados devem ter seguido as mesmas caraterísticas. Os áudios do conjunto de dados para treino, introduzido na secção 2.5.1, são ficheiros em formato mp3 e por isso foi feita a conversão de todos para wav. Depois foi aplicada a separação de fontes, passando de 900 ficheiros para 2700 ficheiros de áudio, onde para cada música estão associados três ficheiros que são do acompanhamento, original e voz. Os ficheiros têm um tamanho fixo de 30 segundos, devidamente validados pelos autores do *dataset* (R. E. S. Panda, 2019), portanto não foi necessário segmentar, mas houve necessidade de aplicar a redução da taxa de amostragem e do número de canais. Com todas estas manipulações o conjunto de dados ficou

[64] https://scikit-learn.org/stable/

assim em situação semelhante ao que será utilizado no sistema desenvolvido para ambiente real.

Para o conjunto de dados de treino da lírica, descrito em 2.5.2, não houve qualquer necessidade de aplicar alterações.

3.2. Extração de caraterísticas

Foram utilizados *scripts* semelhantes aos microsserviços AudioFeaturesExtractor e LyricsFeaturesExtractor, descritos na secção 5.10 e 5.7, para extrair caraterísticas dos conjuntos de dados de áudio e lírica respetivamente. As caraterísticas são valores que descrevem a lírica ou o sinal de entrada do áudio e optou-se por obter o máximo de valores possíveis, entre os disponíveis. A quantidade enorme de algoritmos e características disponíveis torna impossível descrever as mesmas neste documento. As mesmas serão futuramente disponibilizadas *online* e podem ser consultadas na documentação das bibliotecas, para o sinal de áudio [https://essentia.upf.edu/algorithms_reference.html] e lírica [https://github.com/hugoredinho/FeatureExtractionSystem].

Para extrair as caraterísticas da lírica foi utilizada a *framework* escolhida na secção 2.6.2, e foram extraídas exatamente 198 caraterísticas que se baseiam na semântica, estrutura e no conteúdo. Estas são extraídas para um ficheiro Excel com formato csv (*Comma-separated values*).

Algumas caraterísticas do áudio são representadas por séries de dados temporais, isto é, sequências de vários valores para um único sinal áudio. Por isso, foi necessário sumariar as mesmas utilizando estatísticas como o valor mínimo, máximo, mediana, média, variância, média da derivada, média da segunda derivada, variância da derivada, variância da segunda derivada, assimetria (*skewness*) e curtose (*kurtosis*), de entre todas as estatísticas possíveis[65]. Isto é feito para as caraterísticas de baixo-nível, de ritmo, da tonalidade e do espetro dos áudios, nomeadamente Gammatone-frequency Cepstral Coefficients (GFCC) e Mel-

[65] https://essentia.upf.edu/reference/std_PoolAggregator.html

Frequency Cepstrum Coefficients, totalizando 4034 caraterísticas extraídas para cada áudio. Algumas características associadas à tonalidade não retornam valores numéricos, estas caraterísticas estão relacionadas com a escala *(scale*) e com notas e acordes (*Pitch class*) e por isso está a ser associado o valor 0 a *minor* e 1 para *major* no caso da escala, e está a ser utilizada a notação para valores inteiros na chave da peça como definido em *Pitch Class*[66]. As caraterísticas dos áudios foram igualmente guardadas em ficheiros csv.

Após a extração de caraterísticas é necessário preparar os dados extraídos para serem utilizados como entrada para os modelos de classificação porque seria mais difícil o modelo reconhecer padrões se existissem dados inválidos. Para evitar isto foram aplicadas duas abordagens distintas que são documentadas nas duas seguintes secções: 1) tratar os dados em falta e, tendo em conta a quantidade muito elevada de características extraídas principalmente para a componente de áudio, optou-se por 2) encontrar métodos para eliminar caraterísticas potencialmente pouco relevantes.

3.3. Tratamento de dados inválidos

Este processo carateriza-se por avaliar a quantidade de valores em falta (por exemplo características que retornaram um erro, valor não numérico ou *null*) para cada característica, isto porque não valia a pena manter caraterísticas que têm um valor inválido para mais de metade das músicas. Os valores inválidos pode ser causados por diversas razões, normalmente ligados à falta de robustez do algoritmo em causa, problemas ou particularidades do sinal de entrada (ex. momentos de silêncio). Dito isto, são apresentadas nas Figura 15 a Figura 18 as caraterísticas que continham valores em falta/inválidos, quer sejam valores nulos ou infinitos, para cada fonte musical. Como é possível verificar, no áudio é sempre o mesmo tipo de característica de baixo nível que apresenta valores em falta para a estatística curtose (*kurtosis*). Avaliando todos os resultados, foram apenas eliminadas as caraterísticas da componente vocal que apresentavam mais de 50% dos dados em falta.

[66] https://en.wikipedia.org/wiki/Pitch_class

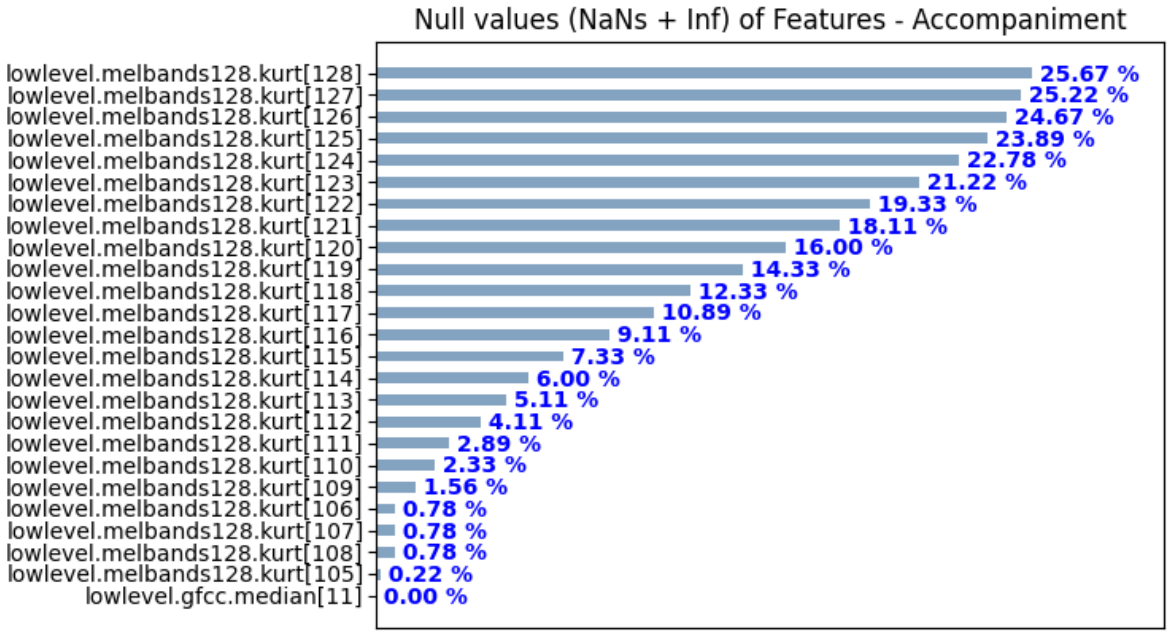


Figura 15 - Dados em falta para caraterísticas do componente de acompanhamento

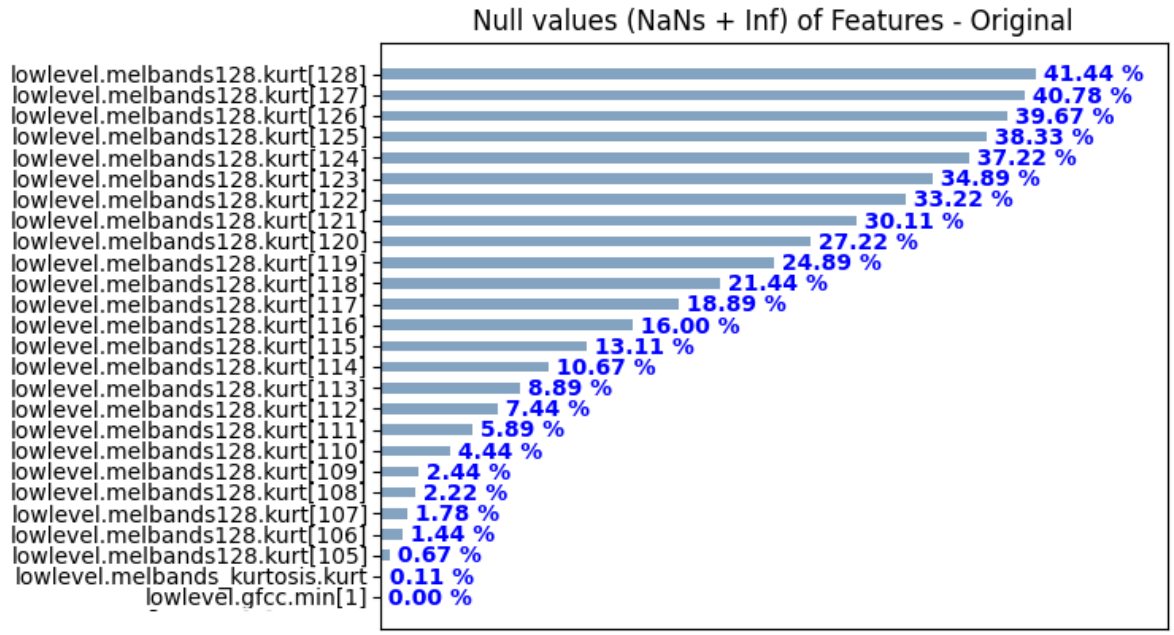


Figura 16 - Dados em falta para caraterísticas do áudio original

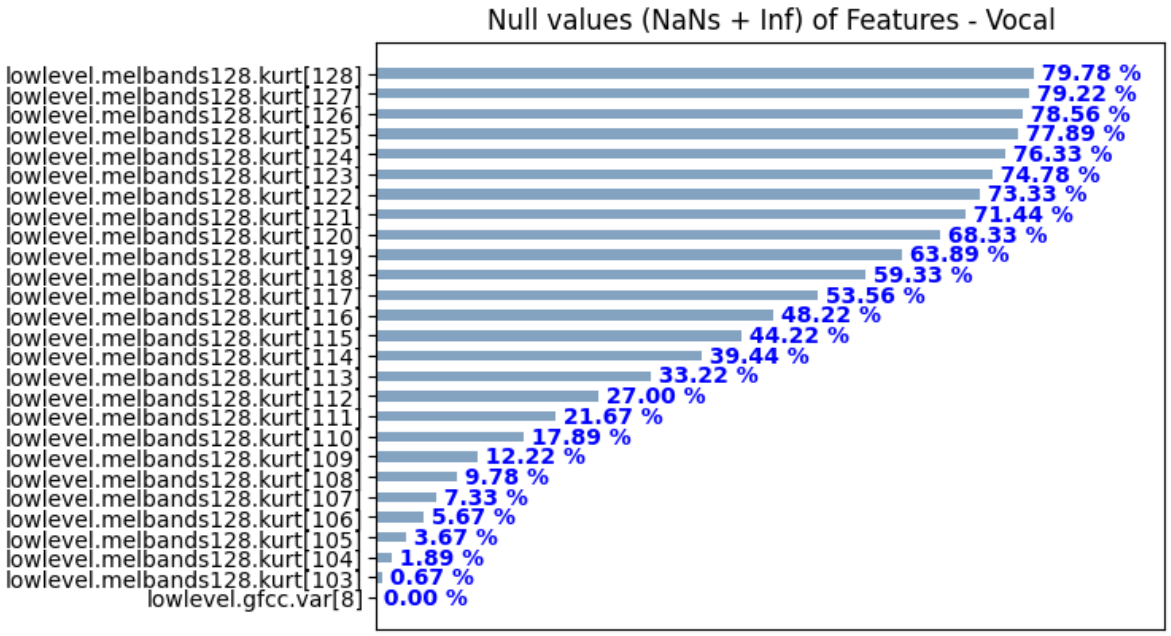


Figura 17 - Dados em falta para caraterísticas da componente vocal

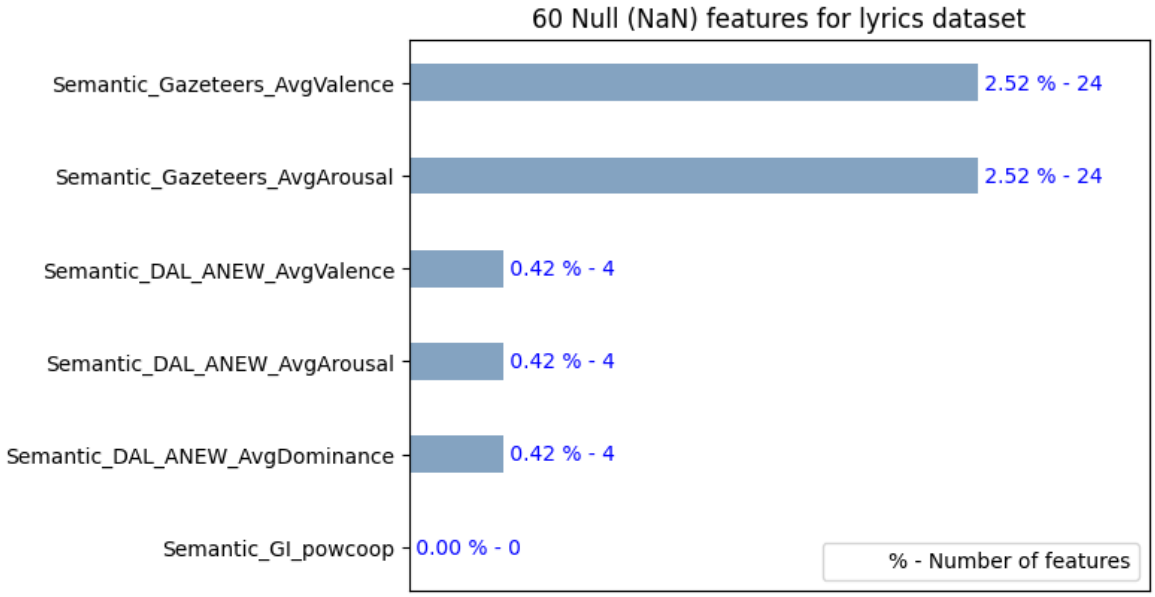


Figura 18 - Dados em falta para caraterísticas da lírica

De seguida, foi ainda necessário substituir os valores em falta para as características remanescentes. Neste caso, os dois valores estatísticos que pareceram os mais acertados para substituir estes valores foram a média ou mediana, pelo que se optou pela utilização da mediana por ser mais robusta a valores fora do padrão (*outliers*), substituindo assim os

valores em falta pela mediana de cada caraterística respetivamente. Houve ainda um número reduzido de excertos do conjunto de dados de áudio que continham apenas componente instrumental e por isso não foi possível obter as caraterísticas da voz para os mesmos, pelo que se optou em vez de descartar as músicas do conjunto, utilizar o valor estatístico mínimo de cada caraterística para as mesmas.

3.4. Eliminação de caraterísticas

Este processo consistiu em eliminar as caraterísticas potencialmente pouco relevantes para o problema de classificação em causa. Isto serve para facilitar a computação no treino dos modelos, aumentando a velocidade de treino e reduzindo a complexidade do classificador. Os dois métodos utilizados foram: 1) avaliar a variância das caraterísticas, eliminando as que não tenham variação (valores semelhantes para todas as músicas) e 2) criar uma matriz de correlação entre pares de características com as restantes, eliminando uma delas.

Foi utilizada a medida de dispersão estatística intervalo interquartil (IQR, Interquartile Range) para remover temporariamente os *outliers*, calculando o intervalo interquartil, ou seja, a diferença entre o terceiro (75%, Q3) e o primeiro (25%, Q1) quartil e depois estabelece o critério de limites inferior e superior com a Equação 1 e Equação 2, respetivamente, onde os valores que ficam além destes limites são considerados *outliers*. Isto foi feito pois na primeira abordagem pretendia-se remover caraterísticas que tivessem os valores todos iguais, sem variância, uma vez que não acrescentam nada de novo ao modelo. No entanto existiam características que tinham os valores todos iguais exceto um ou dois que são considerados *outliers*, portanto esta medida (ignorar *outliers*) permitiu remover caraterísticas que não tinham variância tendo em conta os *outliers*.

$$Q1 - 1{,}5 * IQR$$

Equação 1 - Cálculo do limite inferior do IQR

$$Q3 + 1{,}5 * IQR$$

Equação 2 - Cálculo do limite superior do IQR

Com as caraterísticas que restaram do procedimento de variância foi avaliada a dependência linear entre as caraterísticas utilizando uma matriz de correlação. Esta matriz permite avaliar

a dependência linear entre todos os pares de caraterísticas, ou seja, obter um coeficiente de correlação de *Pearson* para saber quão semelhantes as características são entre elas, porque se duas características forem muito semelhantes, faz apenas sentido manter umas delas. O coeficiente de correlação varia entre os valores -1 e 1, onde quanto mais próximo de -1 ou 1, mais forte é a correlação (inversa ou direta) e, portanto, mais semelhantes as duas características testadas são, podendo eliminar-se uma delas. Quanto mais próximo de 0 é o valor de correlação, menos semelhantes são as duas características comparadas. Está representada na Figura 19 uma matriz de correlação, segundo um mapa de calor (*heatmap*), com algumas características do *dataset* a título de exemplo, onde é possível verificar que as matrizes de correlação são simétricas e por isso só se apresenta o valor para metade da matriz. São testados todos os pares de características e o coeficiente apresentado apenas varia de 0 a 1, pois neste caso foi aplicado o valor absoluto. Após analisar documentação da literatura concluiu-se que não existe um valor fixo para o coeficiente a partir do qual se elimine características, embora habitualmente se elimine uma das características quando o coeficiente é superior a 0,7 ou 0,8. Como neste caso específico o número de características era extenso, pretendia-se eliminar um número maior de caraterísticas, por isso eliminou-se a característica menos significativa do par quando o coeficiente era superior a 0,7.

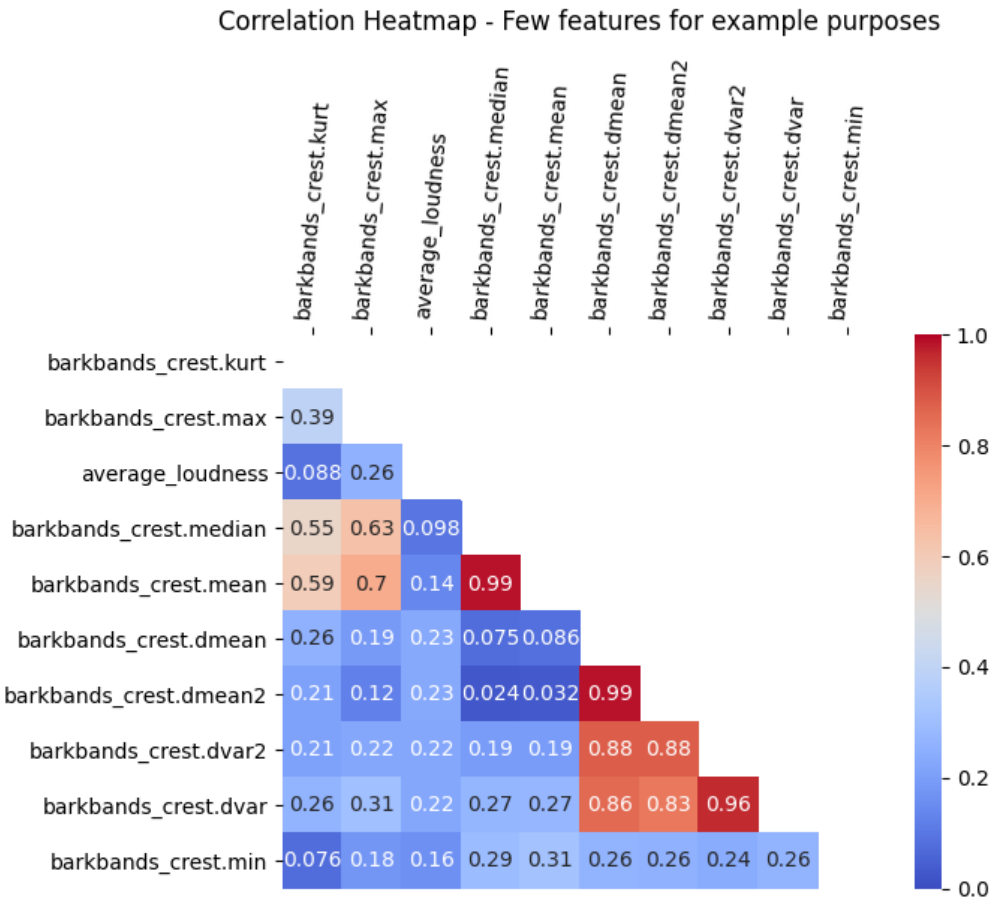


Figura 19 - Excerto de uma matriz de correlação entre pares de características extraídas do áudio

Após a redução das características inválidas ou duplicadas, o número destas foi reduzido para 412 no caso do áudio original, 322 no caso vocal, 396 para o acompanhamento, 972 para o conjunto das 3 fontes áudio e 122 para a lírica. Uma breve descrição destes conjuntos é apresentada de forma gráfica nas Figura 20 a Figura 23. Por questões de espaço apenas é ilustrada informação para o conjunto de dados combinado, que teve como origem 12 102 caraterísticas (4034 caraterísticas para cada uma das 3 fontes) extraídas dos 900 ficheiros de áudio original, 900 de voz, e 900 de acompanhamento (usando a *framework* Essentia), ao qual se seguiu a eliminação e correção de valores inválidos, eliminação de caraterísticas sem variância e posterior eliminação de características semelhantes (via correlação), tal como descrito acima. Para além de ser apresentada por fonte (Figura 20), esta informação é também discriminada por tipo (baixo nível, tonal e ritmo). Nas figuras seguintes é dado mais

detalhe, sendo incluído para cada um destes tipos os vários grupos de características (Figura 21 a Figura 23)

É possível verificar que as características de baixo-nível estão em número muito maior. Isto acontece por serem na maioria métricas de energia (mais fáceis de conceber e mais abstratas do ponto de vista musicológico e longe dos conceitos de alto nível que nós, enquanto humanos, usamos para descrever música e emoção), divididas por bandas e fornecidas em séries temporais, que depois são sumariadas – como é o caso das Mel-frequency cepstral coefficients (MFCCs) - cepstrum de frequência de Mel, uma representação do espectro de potência de um som.

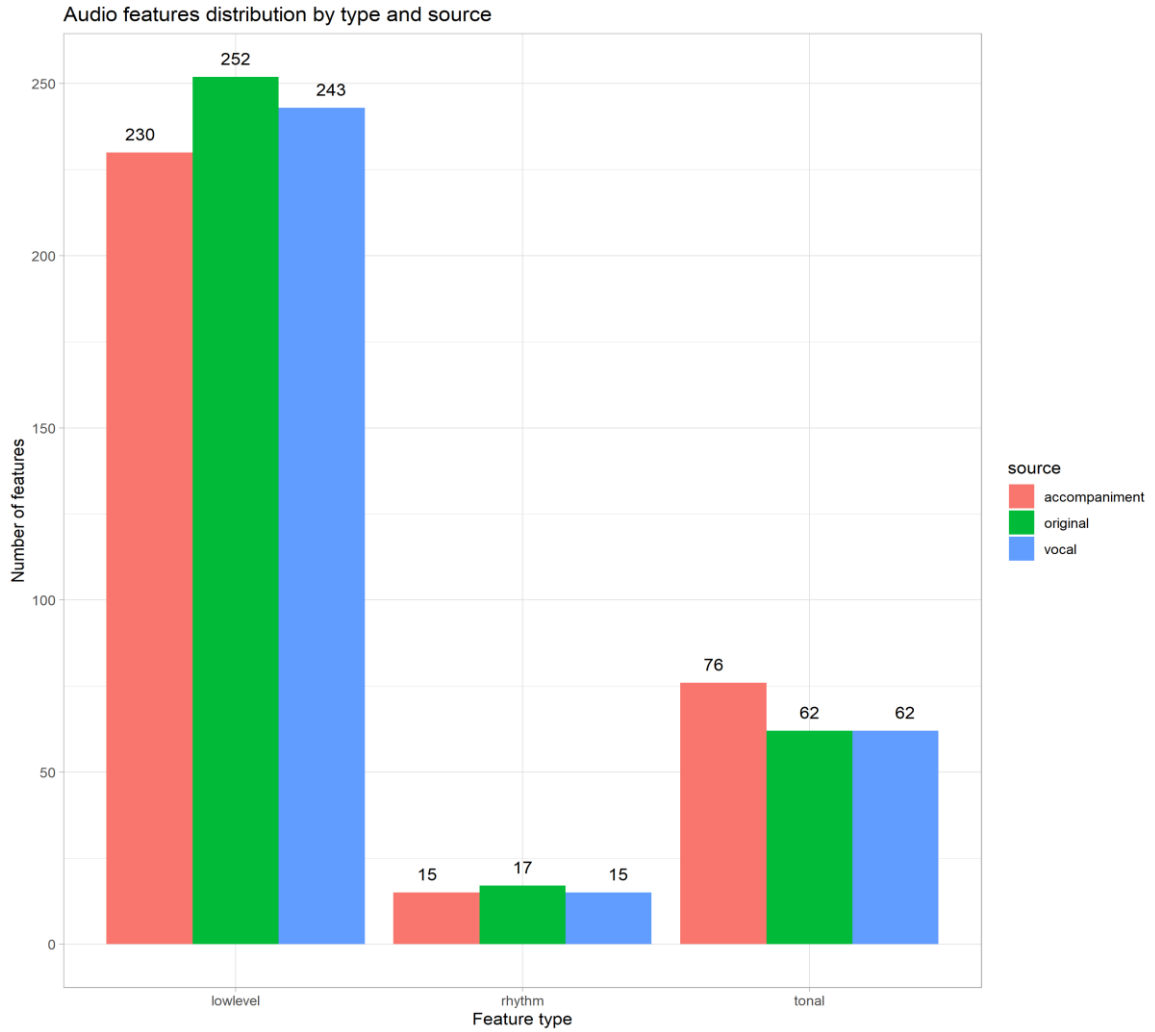


Figura 20 - Distribuição das caraterísticas por tipo e por fonte

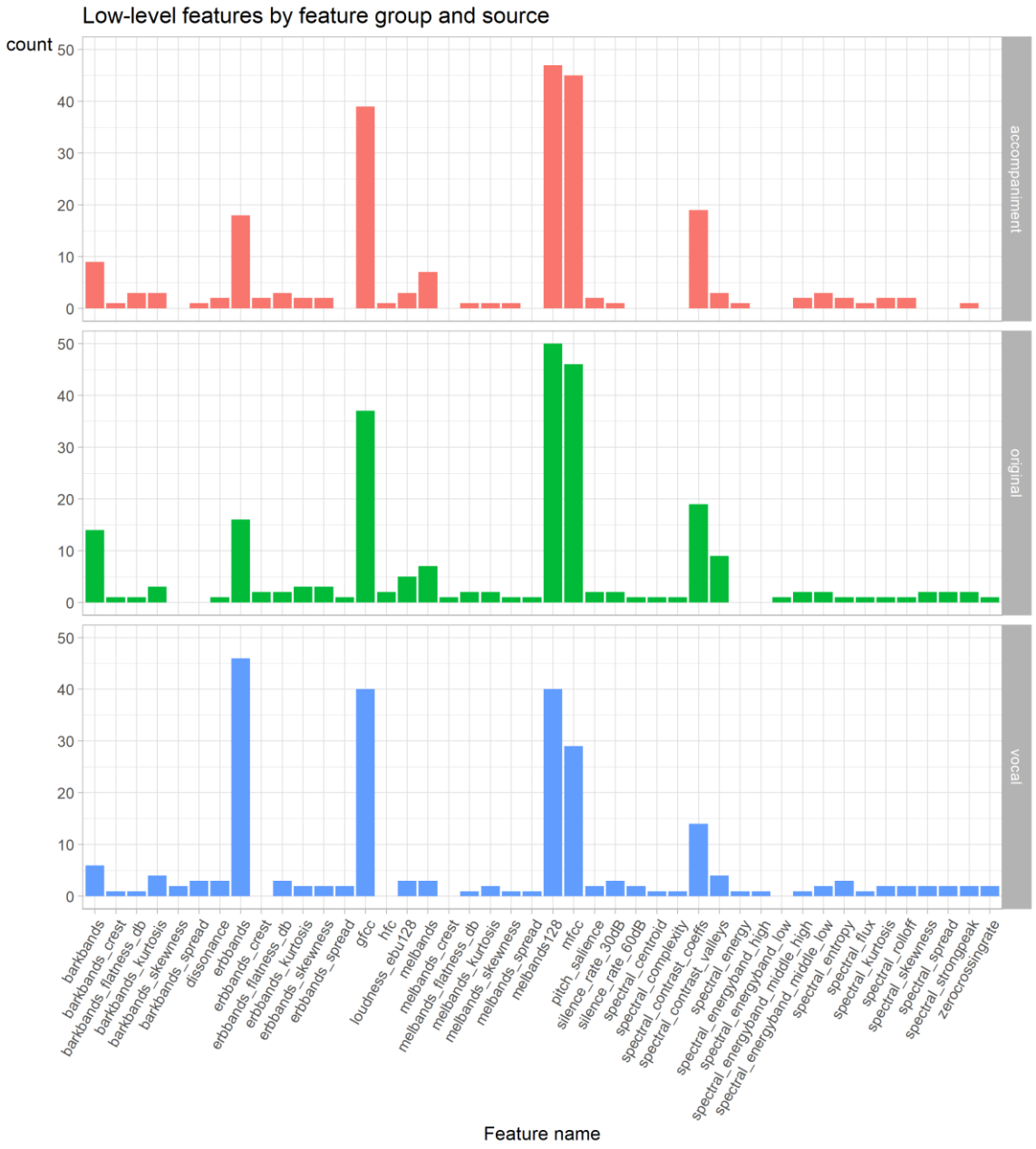


Figura 21 - Caraterísticas de baixo nível discriminadas por grupos de caraterísticas e fonte musical

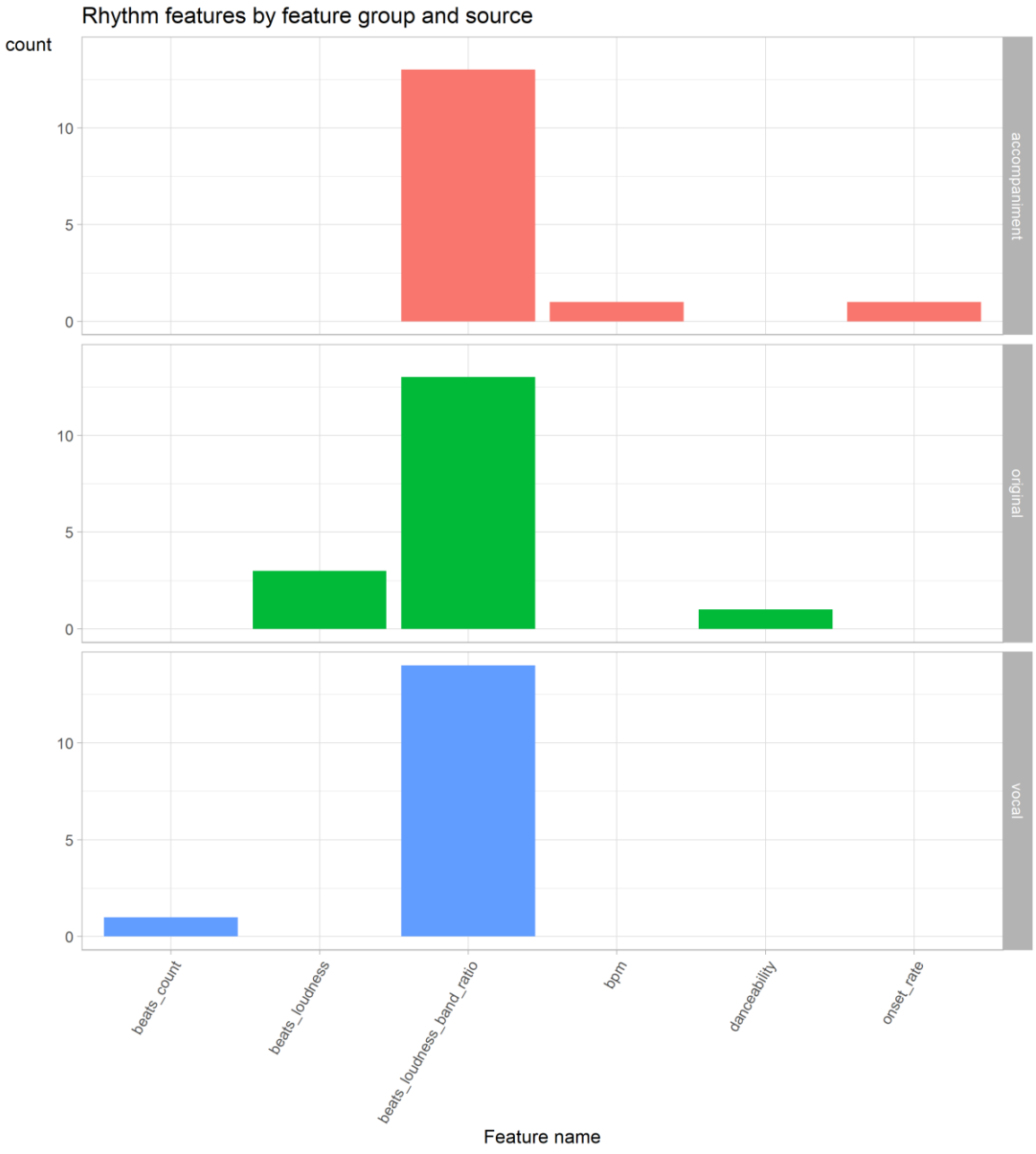


Figura 22 - Caraterísticas de ritmo discriminadas por grupos de caraterísticas e fonte musical

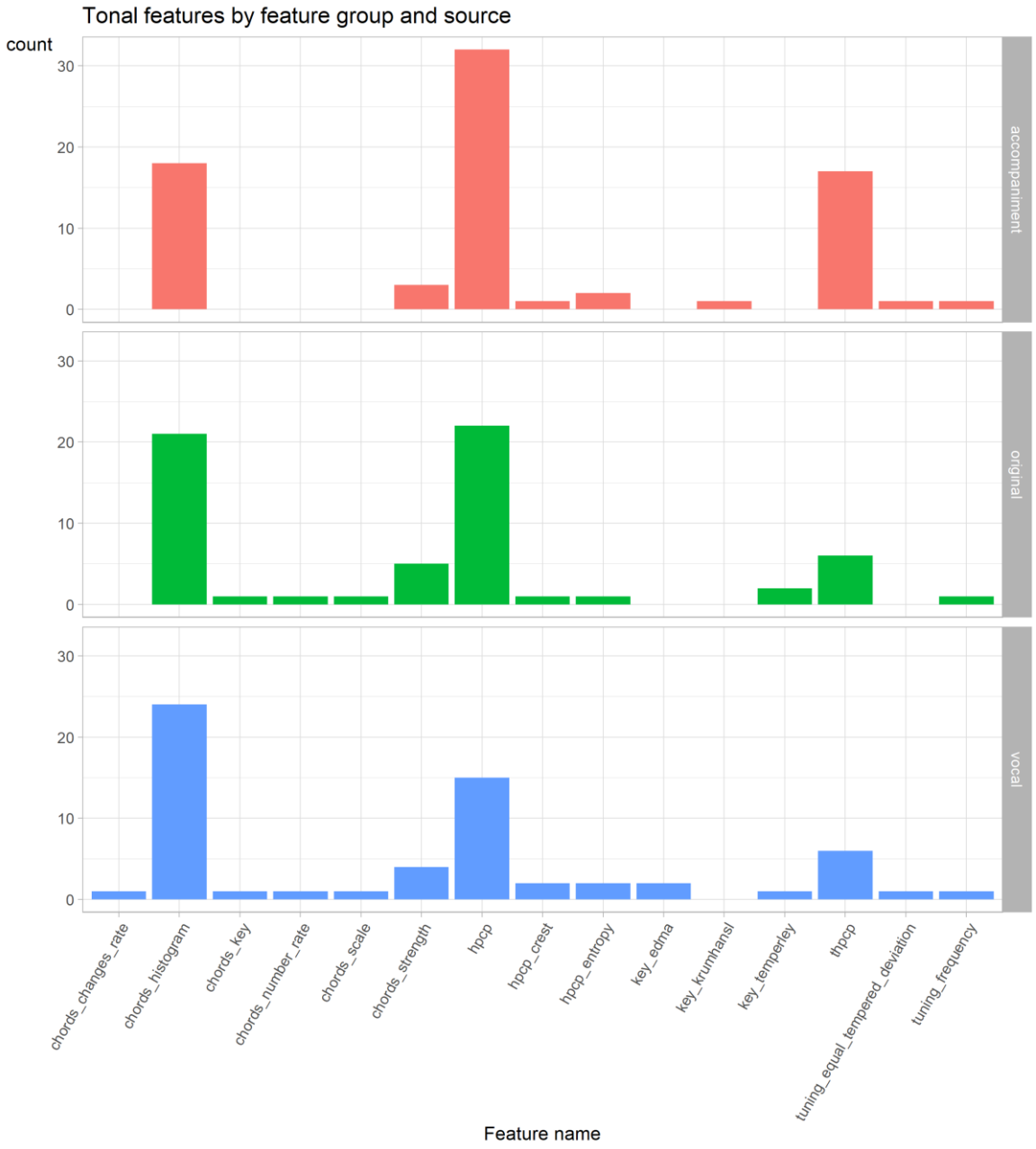


Figura 23 - Caraterísticas de tonal discriminadas por grupos de caraterísticas e fonte musical

3.5. Seleção de caraterísticas

Neste estágio todas as caraterísticas são potencialmente relevantes, tendo em conta que foram eliminadas as caraterísticas pouco relevantes. Mas existem algumas caraterísticas que podem ser mais significativas que outras. Para saber quais as características a utilizar foram

usados dois algoritmos de *ranking* para atribuir um peso a cada caraterística e conseguir ordenar as características por importância.

Os algoritmos de *ranking* utilizados foram Chi^2 (Chi square) e TuRF (Turned ReliefF) que atribuem pesos às características e depois permite que as ordenemos por esses pesos. Chi^2 mede como um modelo se compara aos dados reais observados[67] e TuRF é uma abordagem de eliminação recursiva, onde a cada iteração elimina as características com pontuação mais baixa[68]. Estes dois algoritmos foram utilizados para ordenar as características por importância, de onde resultaram dois *rankings* para cada conjunto de dados a serem utilizados posteriormente para treinar/testar os modelos de classificação. Esta estratégia permite ter uma base confiável para selecionar as caraterísticas a utilizar, em vez de ser uma decisão completamente aleatória.

3.6. Aprendizagem computacional

À semelhança do que havia sido feito na prova de conceito inicial, o algoritmo de aprendizagem computacional SVM foi mantido como justificado na secção 2.7, assim como a explicação dos parâmetros. Os parâmetros que permitem ter um melhor desempenho variam de conjunto para conjunto de dados, não existindo uma regra clara para a definição dos mesmos. Por outro lado, se a procura pelos melhores parâmetros for feita manualmente é bastante aborrecida e extremamente demorada, uma vez que é necessário treinar e testar modelos com diferentes parâmetros para avaliar os seus desempenhos. Por esse motivo foi o utilizado o método GridSearchCV[69] que faz uma pesquisa exaustiva sobre um conjunto de parâmetros que se pretende avaliar, treinando e testando modelos e apresentando os resultados. Por questões de robustez dos resultados, foi também utilizada uma estratégia de validação cruzada (*cross validation*), neste caso RepeatedStratifiedKFold[70] com $k = 10$, $r = 10$, que divide o conjunto de dados em vários subconjuntos denominados *folds*. A estratégia adotada divide o conjunto de dados em 10 subconjuntos ou *folds* (*10-fold*) com uma

[67] https://www.investopedia.com/terms/c/chi-square-statistic.asp
[68] https://www.sciencedirect.com/science/article/pii/S1532046418301400
[69] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[70] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html

distribuição equilibrada entre quadrantes (*stratified*), fazendo 10 testes de classificação, pegando sempre num *fold* diferente para testar e os restantes 9 para treinar. No fim este processo é repetido 10 vezes (*repeated*). Embora muito mais demorado, este processo é mais robusto pois fornece resultados com base em 100 modelos treinados com conjuntos distintos.

```
{
    'kernel': ['linear'],
    'C': [0.1, 1, 10, 100, 1000]
},
{
    'kernel': ['poly'],
    'degree': [2, 3, 4],
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'scale', 'auto']
},
{
    'kernel': ['rbf'],
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'scale', 'auto']
},
{
    'kernel': ['sigmoid'],
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'scale', 'auto']
}
```

Figura 24 - Parâmetros testados para classificadores

Os parâmetros testados estão representados acima na Figura 24 e já tinham sido explicados na secção 2.7. Como ainda assim o processo de procura por essa combinação de parâmetros é exaustivo porque se pretende ter cinco classificadores, cada um tem as suas caraterísticas e os parâmetros podem variar entre eles, este método foi apenas aplicado a um conjunto de características que pareciam ter sentido, até porque o processo era todo duplicado visto que está a ser feito para ambos os *ranking* de caraterísticas obtidos com o processo explicado na secção 3.5.

O processo de pesquisa exaustiva foi assim aplicado para as melhores 10, 25, 50, 100 e depois de 100 em 100 até ao fim das caraterísticas para cada um dos cinco conjuntos de dados com ambos os algoritmos de *ranking*, onde foram obtidos diversos modelos treinados (10 *folds* * 10 repetições) e respetivas métricas para cada combinação de parâmetros de cada número de características, devolvendo a média das métricas.

Para avaliar o desempenho dos parâmetros foi considerada a métrica F1-score que é bastante utilizada para estes fins, consistindo numa média entre as métricas de *precision* e *recall*, onde a precisão mede a relevância do resultado enquanto o *recall* mede quantos resultados verdadeiramente relevantes são retornados (fórmulas exibidas da Equação 3 a Equação 5). Em "macro" F1 é calculada uma pontuação F1 separada para cada classe e depois calculada a média.

$$F1\ score = 2 * \frac{precision * recall}{precision + recall}$$

Equação 3 - Cálculo da métrica F1-score

$$precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

Equação 4 - Cálculo da métrica *precision*

$$recall = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Negatives\ (FN)}$$

Equação 5 - Cálculo da métrica *recall*

São apresentados vários gráficos, da Figura 28 a Figura 32, ilustrando o desempenho de cada um dos *kernels*, onde foi considerada a melhor combinação de parâmetros de cada *kernel* para cada número de características testado. Em todos estes, à esquerda é mostrado o resultado usando o ranking de características dado pelo método Chi^2, enquanto o da direita utiliza TuRF.

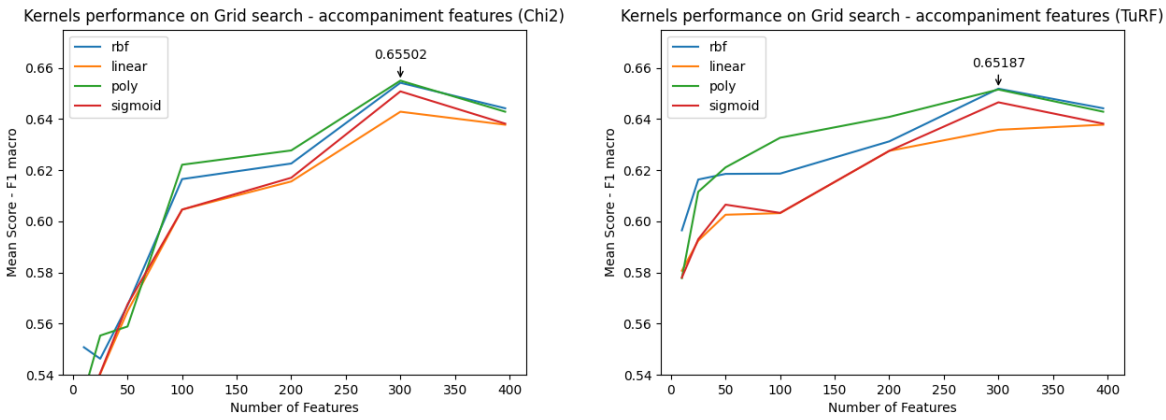


Figura 25 - Desempenho dos *kernels* para as caraterísticas de acompanhamento

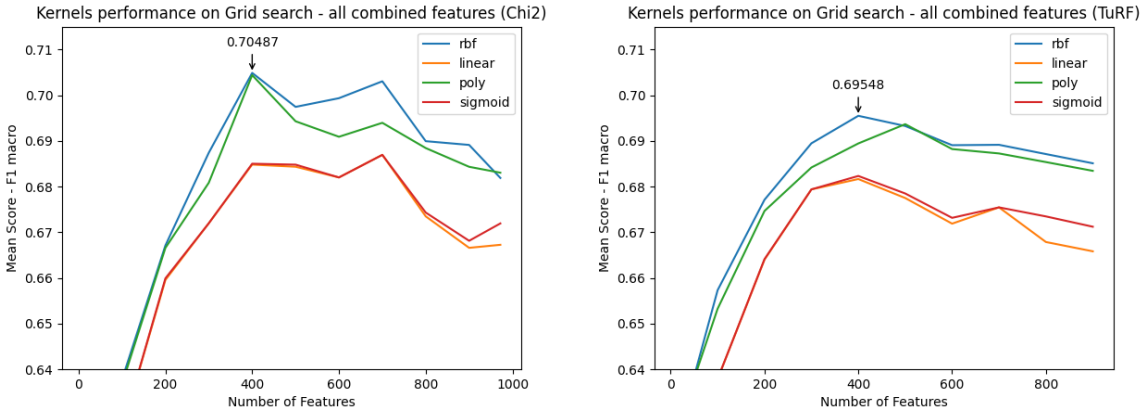


Figura 26 - Desempenho dos *kernels* para todas as caraterísticas de áudio combinadas

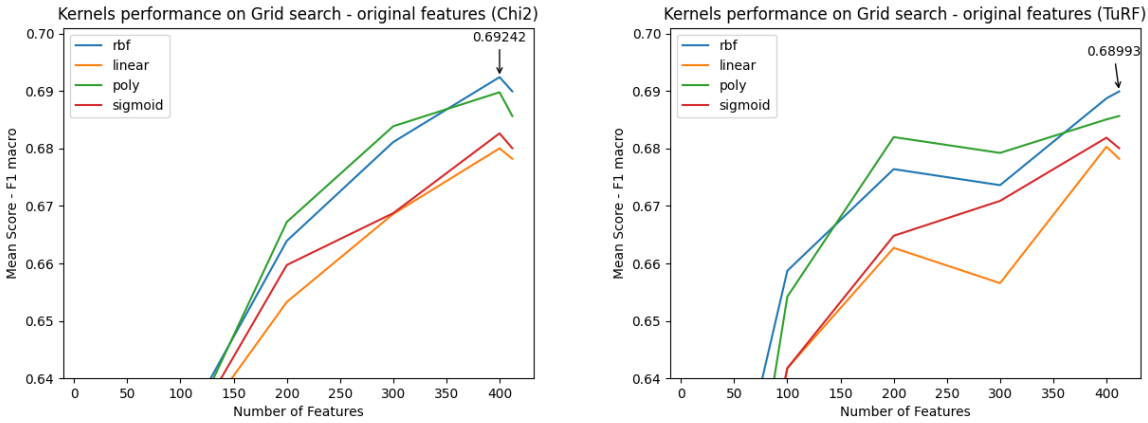


Figura 27 - Desempenho dos *kernels* para as características do áudio original

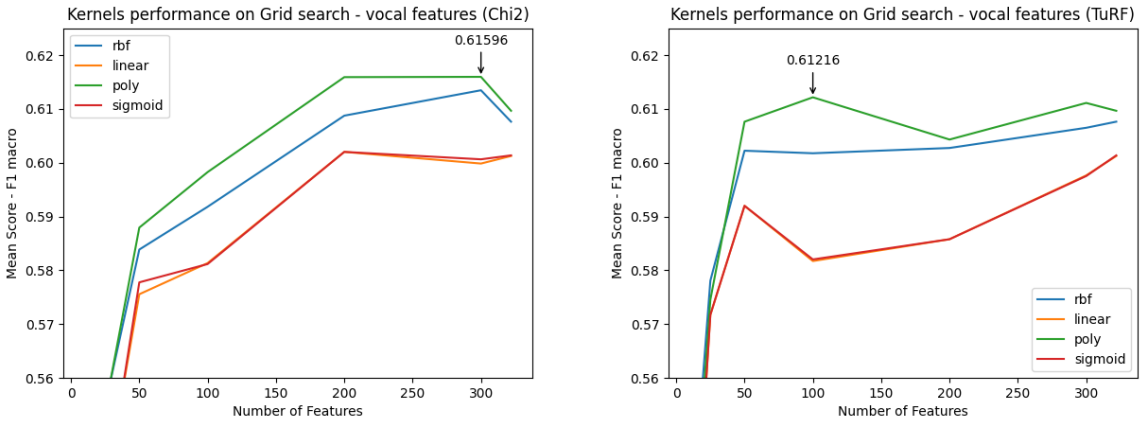


Figura 28 - Desempenho dos *kernels* para as caraterísticas vocais

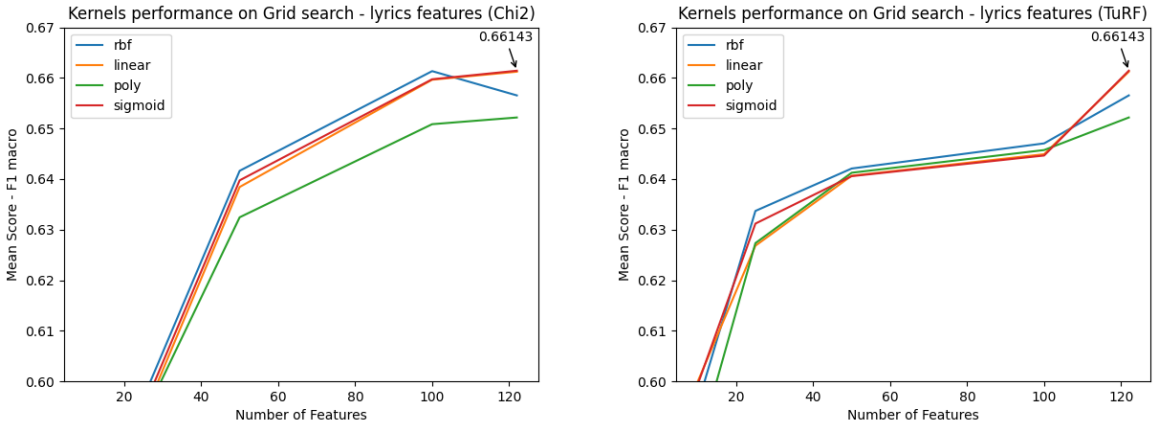


Figura 29 - Desempenho dos *kernels* para as caraterísticas da lírica

O objetivo de todo este processo foi obter os melhores parâmetros para cada um dos cinco conjuntos de dados para depois se utilizar apenas esses parâmetros, e não ser uma computação tão pesada, para treinar e testar modelos com todas as melhores características. Como foi possível verificar, os *kernels* que tiveram melhor desempenho foram *poly* para acompanhamento e voz (Figura 25 e Figura 28), *rbf* para todas as caraterísticas de áudio combinadas e original (Figura 26 e Figura 27), e o *sigmoid* para a lírica (Figura 29). Estes resultados permitiram identificar a melhor *kernel* (entre as quatro testadas) para cada um dos problemas. Com base nisto, foram então feitos novos testes utilizando apenas a melhor kernel para cada problema, mas agora testando os melhores parâmetros desses mesmos *kernels* para os respetivos conjuntos de dados. Os resultados estão ilustrados nas figuras seguintes (Figura 30 a Figura 34).

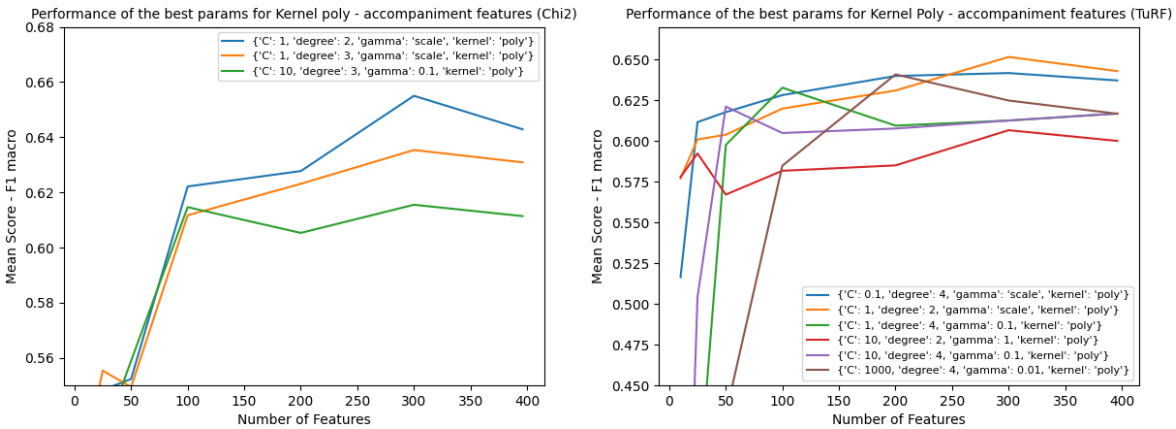


Figura 30 - Desempenho dos melhores parâmetros do *kernel* poly para caraterísticas de acompanhamento

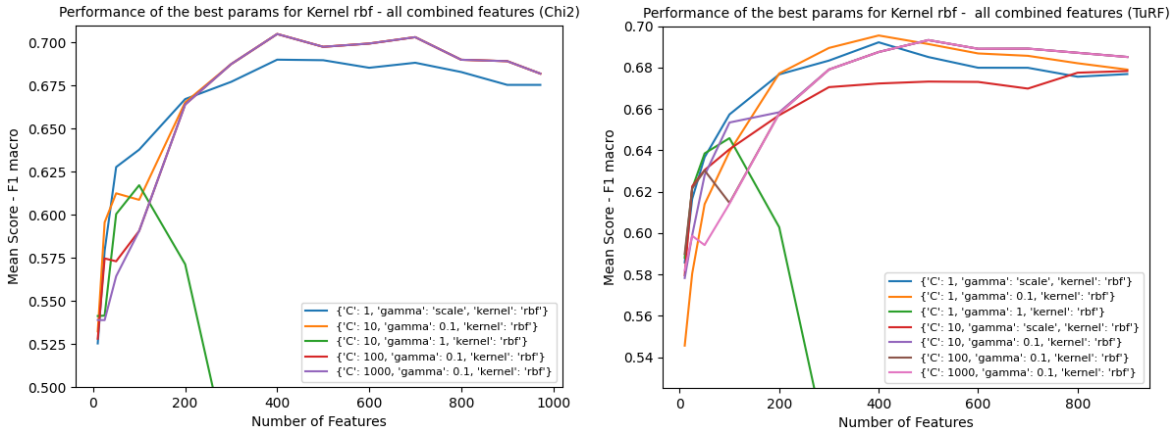


Figura 31 - Desempenho dos melhores parâmetros do *kernel* rbf para todas as caraterísticas de áudio

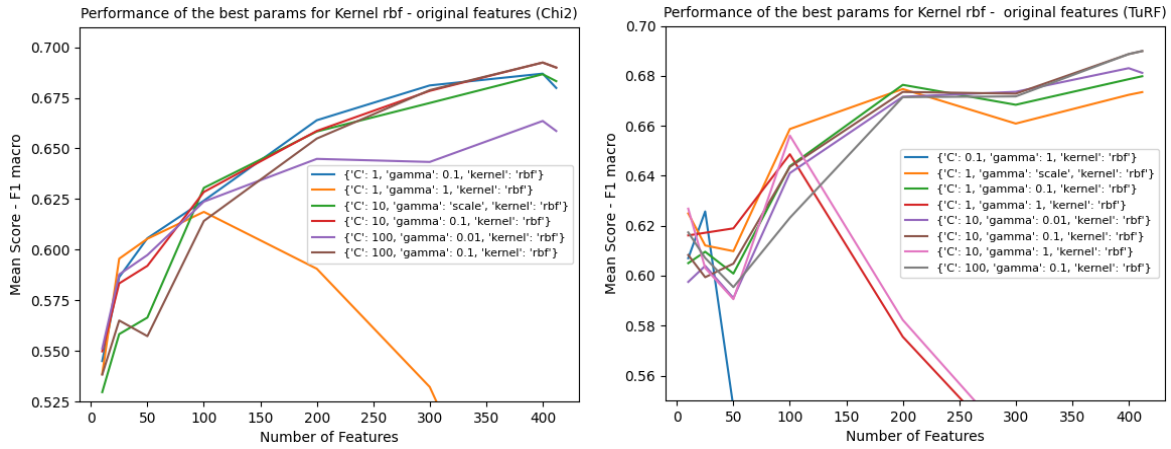


Figura 32 - Desempenho dos melhores parâmetros do *kernel* rbf para caraterísticas do áudio original

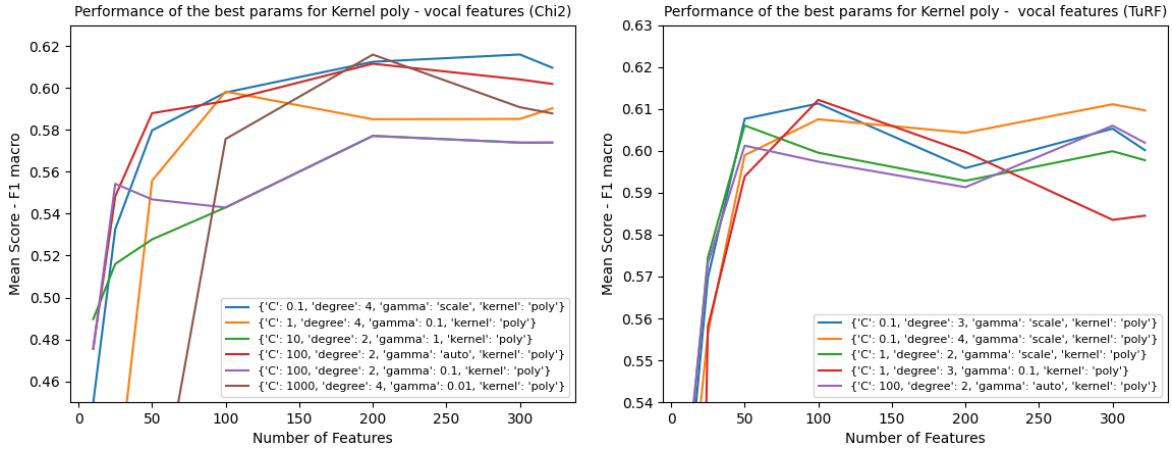


Figura 33 - Desempenho dos melhores parâmetros do *kernel* poly para caraterísticas da voz

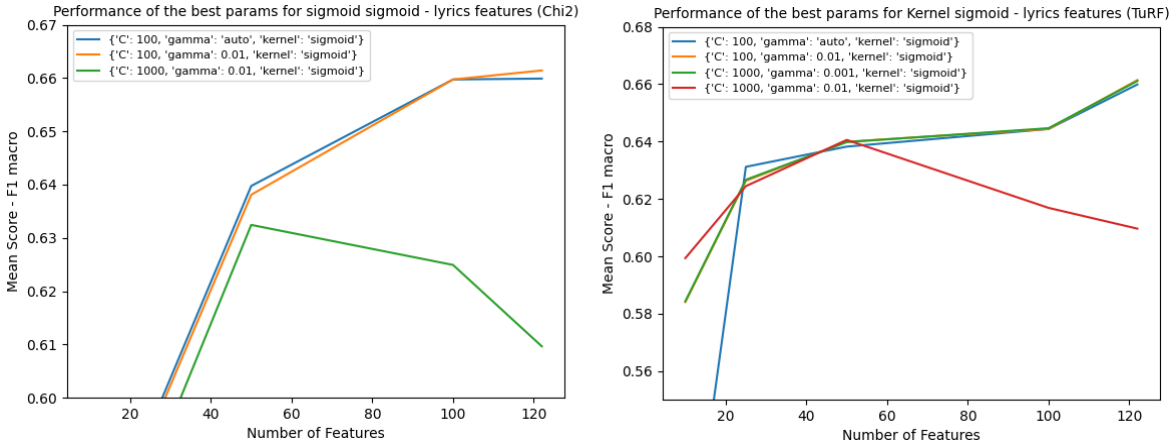


Figura 34 - Desempenho dos melhores parâmetros do *kernel* sigmoid para caraterísticas da lírica

Este processo permitiu identificar a melhor combinação de parâmetros para cada problema/conjunto de dados. Ainda assim, os dados obtidos não permitem saber o número ideal de características, uma vez que foram testados intervalos grandes (as melhores 10 características, as melhores 25, 50, 100 e depois de 100 em 100). Por esta razão foi realizado o procedimento de validação cruzada para a combinação de todas as características por ordem de peso, obtido através dos métodos Chi^2 e TuRF. Ou seja, treinar e testar 10 vezes (10-fold) para cada um destes algoritmos de *ranking*, um modelo com a melhor caraterística (top1), um modelo com as duas melhores caraterísticas (top2), com as três e assim por diante (até topN), retornando sempre um conjunto de métricas. Foi utilizada novamente a métrica de F1-score para avaliar qual o número de características que permitiu obter um melhor resultado, como mostram os gráficos de seguida (Figura 35 a Figura 39).

Todas estas opções foram tomadas por questões de tempo e recursos computacionais disponíveis. Num cenário ideal seria feito o teste exaustivo de todas as *kernels* vs. parâmetros vs. combinações de características.

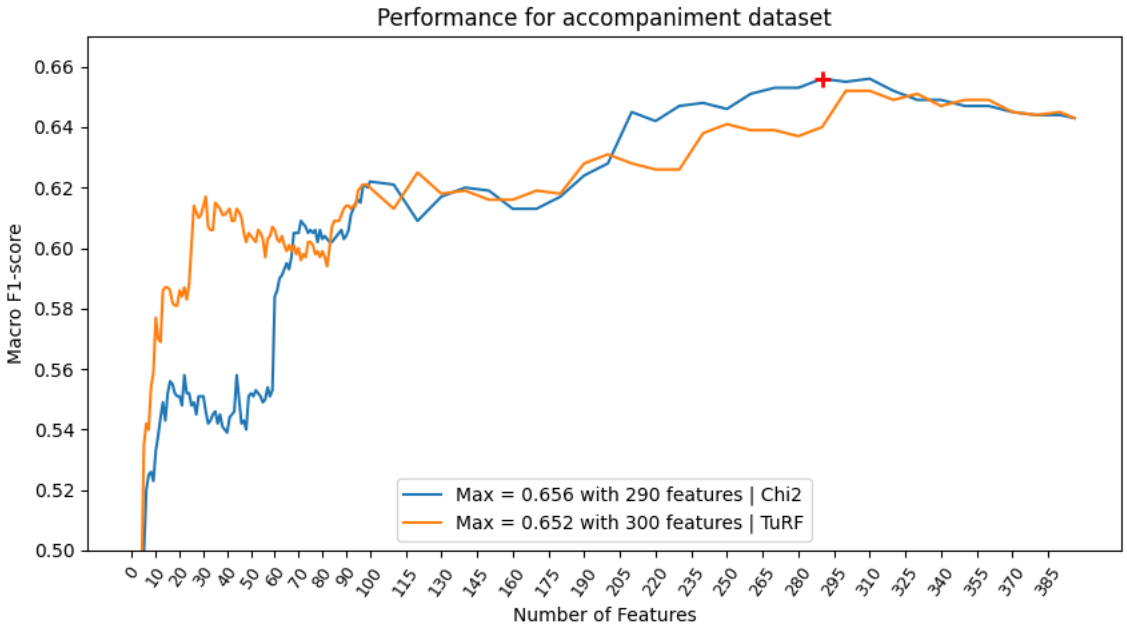


Figura 35 - Métrica F1-score para todos os modelos treinados das caraterísticas de acompanhamento

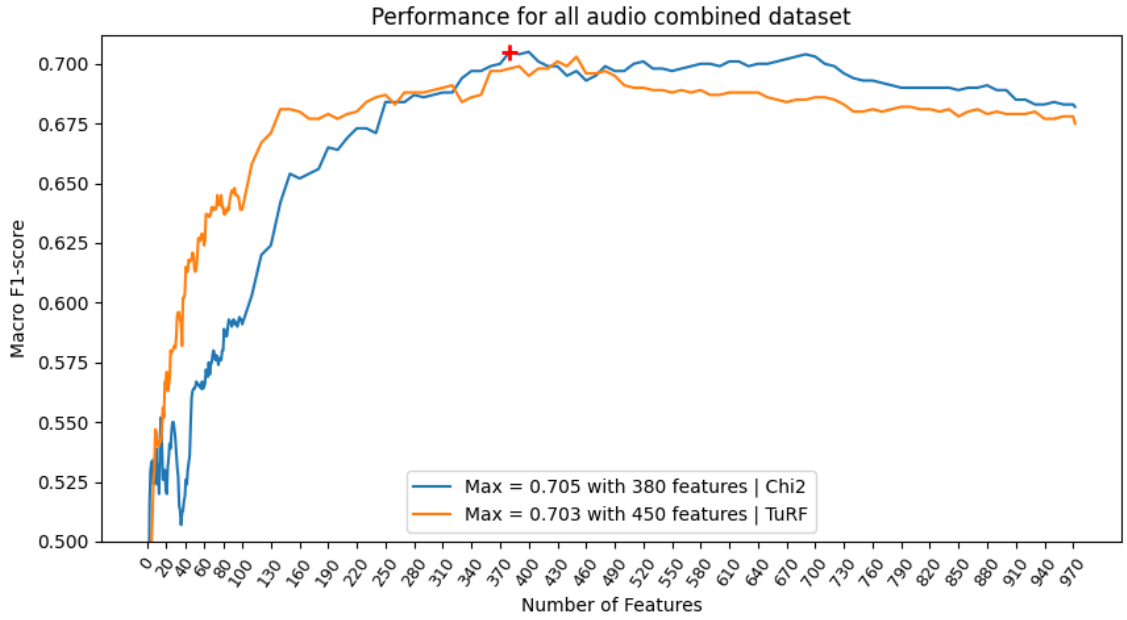


Figura 36 - Métrica F1-score para todos os modelos treinados das caraterísticas de áudio combinadas

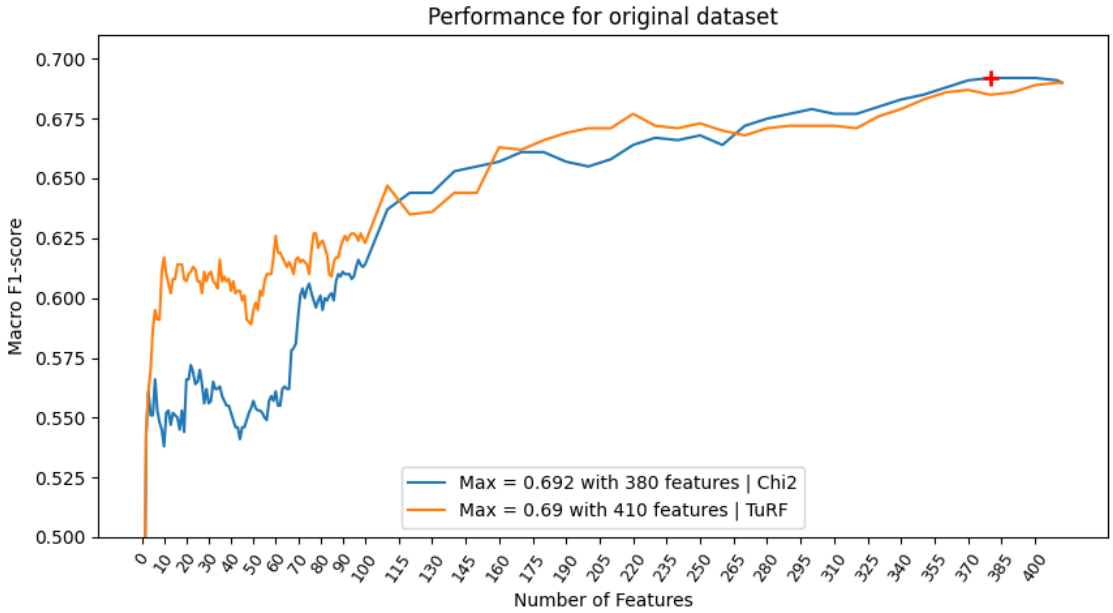


Figura 37 - Métrica F1-score para todos os modelos treinados das caraterísticas do áudio original

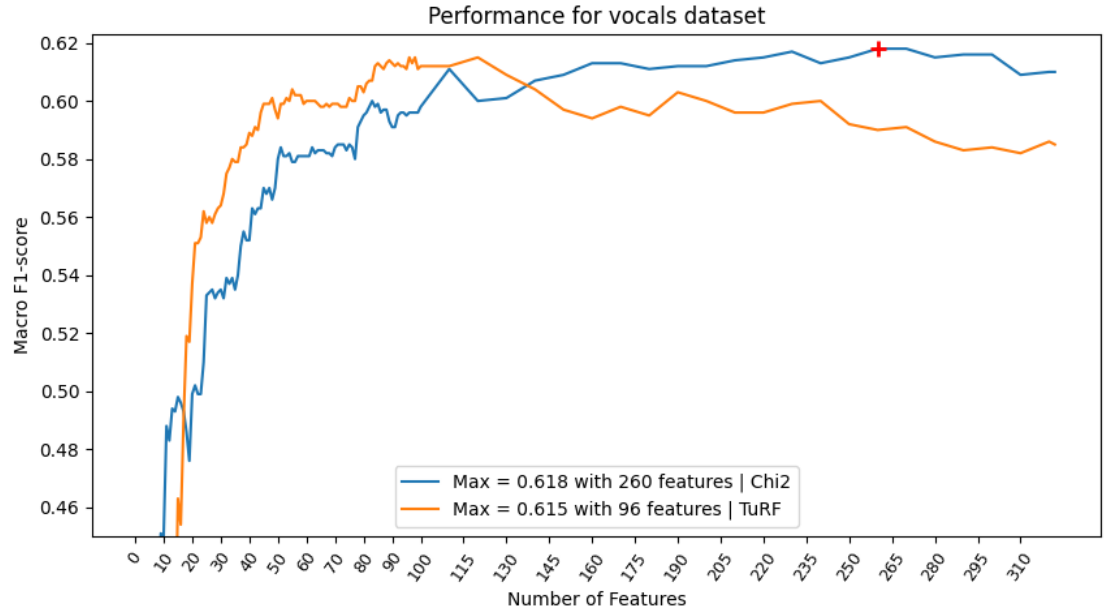


Figura 38 - Métrica F1-score para todos os modelos treinados das caraterísticas da voz

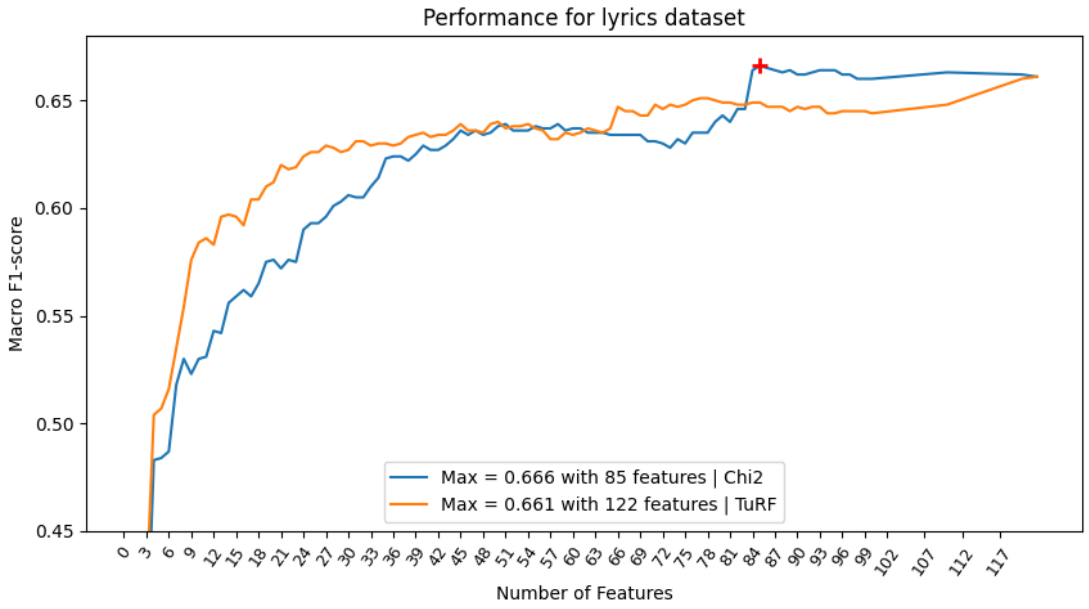


Figura 39 - Métrica F1-score para todos os modelos treinados das caraterísticas da lírica

3.7. Resultados

A realização dos vários testes descritos nas seções anteriores permitiram identificar as várias combinações de características, classificadores e parâmetros, que melhores resultados obtêm para cada um dos cinco problemas em estudo. A Tabela 6, apresenta um resumo dos resultados obtidos para as melhores combinações de parâmetros e *ranking*. Os modelos que utilizaram o algoritmo de *ranking* Chi^2 obtiveram na generalidade valores de F1-measure mais elevados, sendo a diferença para o algoritmo TuRF baixa dado o mesmo número de características.

De acordo com a Tabela 6, é possível concluir que o modelo que permite obter uma melhor taxa de acertos entre os cinco classificadores treinados é aquele que utiliza todas as fontes áudio (acompanhamento, original e voz), atingindo 70,5% ± 0,049 de F1-score utilizando 380 características e SVM com *kernel rbf*, custo 1000 e *gamma* de 0,1. O classificador utilizando áudio original fica bastante próximo com 69,2% ± 0,052 e também 380 características. O modelo com pior desempenho foi o que utiliza apenas as caraterísticas extraídas do áudio da voz isolada. Este resultado acaba por ser esperado, uma vez que ao isolar apenas a voz, perdendo todo o acompanhamento, perde-se grande parte da informação

musical que foi usada pelos voluntários para anotar o conjunto de dados, ficando apenas a possível informação emocional que a voz do cantor ou cantora carrega. Uma outra possível justificação para tal é o facto de haver alguns áudios apenas com componente instrumental, para os quais por razões óbvias não foi possível obter as caraterísticas da parte vocal. Neste caso, estas foram substituídas pelo valor mínimo de cada caraterística, o que pode estar longe do valor que mais sentido faria e assim influenciar o resultado negativamente.

Dataset	**Algorítmo de Ranking**	**Parâmetros**	**Número de *Features***	**Macro F1-score**	***Precision***	***Recall***	**95% do máximo macro F1-score**	**99% do máximo macro F1-score**
Acompanhamento	CHI^2	Cost: 1 Degree: 2 Gamma: scale Kernel: poly	290	0,656 ± 0,051	0,665 ± 0,051	0,657 ± 0,050	0,624 ± 0,053 com 190 *features*	0,651 ± 0,048 com 260 *features*
Acompanhamento	TuRF	Cost: 1 Degree: 2 Gamma: scale Kernel: poly	300	0,652 ± 0,050	0,660 ± 0.050	0,653 ± 0,049	0,620 ± 0,053 com 96 *features*	0,652 ± 0,05 com 300 *features*
Todas as fontes combinadas	CHI^2	Cost: 1000 Gamma: 0,1 Kernel: rbf	380	0,705 ± 0,049	0,712 ± 0,049	0,706 ± 0,049	0,673 ± 0,044 com 220 *features*	0,699 ± 0,051 com 360 *features*
Todas as fontes combinadas	TuRF	Cost: 1 Gamma: 0,1 Kernel: rbf	450	0,703 ± 0,046	0,712 ± 0,045	0,705 ± 0,045	0,671 ± 0,051 com 130 *features*	0,697 ± 0,045 com 360 *features*
Original	CHI^2	Cost: 100 Gamma: 0,1 Kernel: rbf	380	0,692 ± 0,052	0,699 ± 0,052	0,694 ± 0,052	0,661 ± 0,047 com 170 *features*	0,688 ± 0,051 com 360 *features*

Original	TuRF	Cost: 100 Gamma: 0,1 Kernel: rbf	410	0,690 ± 0,051	0,697 ± 0,052	0,692 ± 0,051	0,603 ± 0,050 com 160 *features*	0,686 ± 0,051 com 360 *features*
Vocal	CHI^2	Cost: 0,1 Degree: 4 Gamma: scale Kernel: poly	260	0,618 ± 0,054	0,627 ± 0,053	0,619 ± 0,054	0,591 ± 0,051 com 78 *features*	0,613 ± 0,055 com 160 *features*
Vocal	TuRF	Cost: 1 Degree: 3 Gamma: 0,1 Kernel: poly	96	0,615 ± 0,053	0,626 ± 0,051	0,617 ± 0,052	0,584 ± 0,054 com 40 *features*	0,609 ± 0,052 com 85 *features*
Lírica	CHI^2	Cost: 100 Gamma: 0,01 Kernel: sigmoid	85	0,666 ± 0,036	0,678 ± 0,038	0,664 ± 0,037	0,636 ± 0,047 com 45 *features*	0,664 ± 0,032 com 84 *features*
Lírica	TuRF	Cost: 100 Gamma: 0,01 Kernel: sigmoid	122	0,661 ± 0,041	0,671 ± 0,043	0,660 ± 0,042	0,629 ± 0,042 com 27 *features*	0,660 ± 0,040 com 120 *features*

Tabela 6 - Melhores combinações de parâmetros para cada dataset e algumas métricas

Para complementar os resultados da Tabela 6, a Figura 40 apresenta a distribuição das melhores 400 características (top400) áudio do conjunto das 3 fontes combinadas, distribuídas por grupo e tipo. Não sendo possível de tirar grandes conclusões a partir desta, é, no entanto, possível ver que no caso da voz, foram selecionadas características ligadas ao ruído, volume e dissonância (*zero crossing rate, loudness, dissonance*), enquanto no sinal original e acompanhamento, estão presentes características ligadas ao ritmo e batida da música (*beats_loudness, danceability, onset_rate*). Para todas as fontes existe uma grande quantidade de características de baixo nível, abstratas, que descrevem propriedades do sinal. Algo que já tinha sido observado no trabalho que serviu de base a estes testes (R. Panda et al., 2020b). No futuro será interessante tentar introduzir novas características de alto nível no sistema, nomeadamente algumas das identificadas como relevantes para MER (R. Panda et al., 2020a).

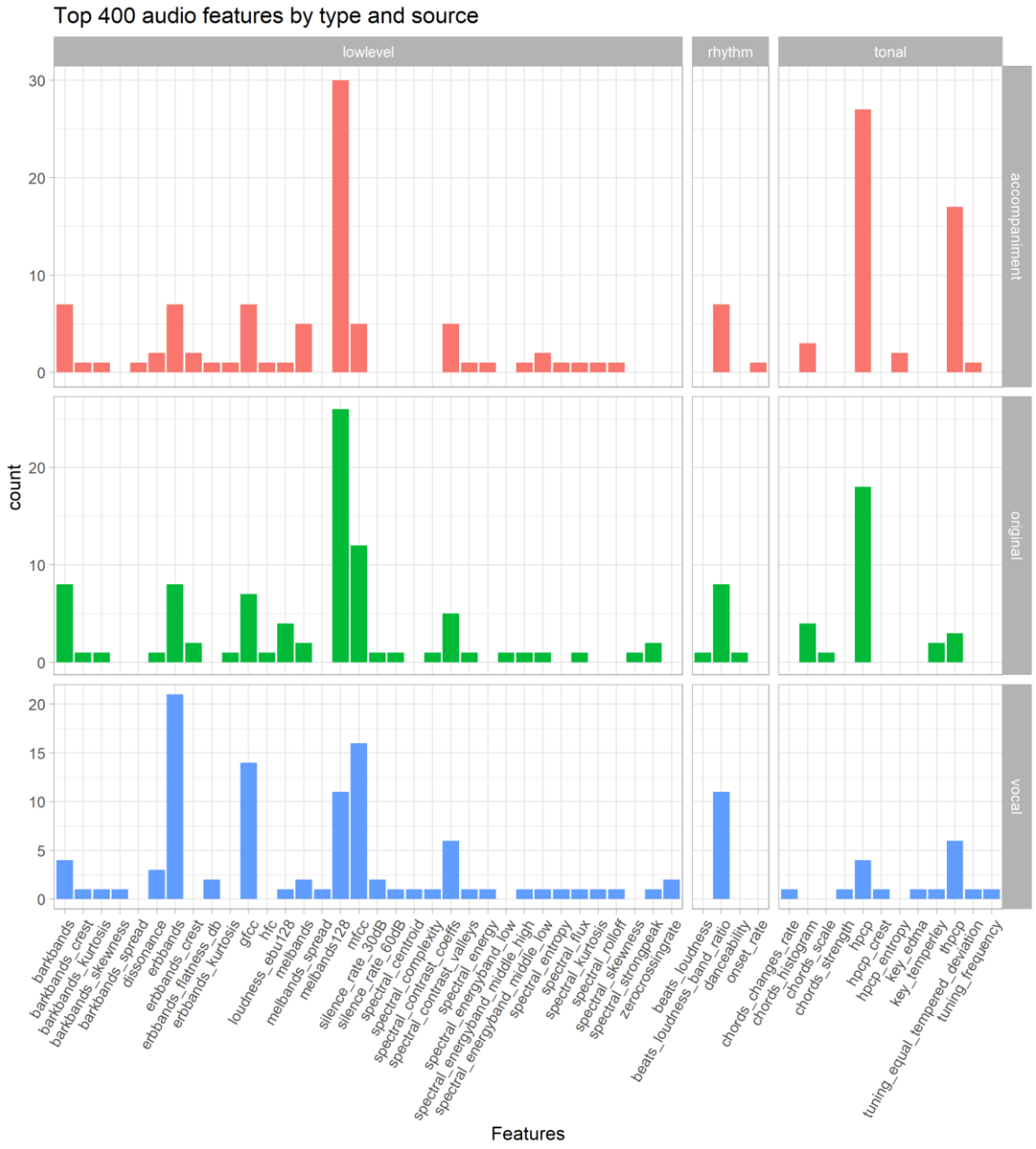


Figura 40 - Distribuição das melhores 400 características (top400) áudio do conjunto das 3 fontes combinadas, distribuídas por grupo e tipo

3.7.1. Criação dos modelos de classificação para produção

O conhecimento sobre o conjunto de características, classificadores e parâmetros que permitem obter os melhores resultados para cada problema serve de base aos classificadores a usar em produção. Para cada um dos classificadores a treinar é necessário:

1. Selecionar as características desejadas (com base no *ranking* e resultados) e guardar o mesmo (*ranking*);
2. Normalizar o valor dessas características usando o *scaler* pretendido;
3. Guardar o *scaler* para ficheiro, de forma que se possa normalizar as caraterísticas em ambiente real para a mesma escala;
4. Treinar o modelo, passando-lhe as características normalizadas e as respetivas anotações;
5. Guardar em ficheiro o modelo de classificação treinado.

Estes modelos de classificação e ficheiros de normalização são depois utilizados no sistema, fazendo parte de um dos microsserviços de classificação.

O sistema de classificação de vídeos do YouTube utilizará estes modelos em tempo real:

1. Os microsserviços LyricsFeaturesExtractor e AudioFeaturesExtractor extraem todas as características possíveis de um novo vídeo;
2. São selecionadas as desejadas com base no *ranking* previamente guardado;
3. As mesmas são normalizadas com base no respetivo *scaler*;
4. O modelo de classificação é carregado e é feita a previsão/classificação emocional do novo exemplo.

Para além dos cinco modelos treinados, foram realizados ainda alguns testes para avaliar como os modelos se comportavam, e para exemplo está na Figura 41 uma matriz de confusão para um modelo que classifica o acompanhamento, onde foi utilizado 90% do conjunto de dados para treinar o modelo e 10% para testar, de onde resultou então esta matriz de confusão. Na matriz da esquerda encontra-se o número de vezes em que ocorreram as classes, por exemplo existiam 25 músicas de teste que tinham a emoção real Alegre e o modelo previu 23 vezes como alegre e 2 vezes como Tensa. Na esquerda encontra-se a frequência de cada classe onde as somas dos valores de cada linha resultam em 1, ou seja, se existiam 25 músicas alegres e o modelo acertou em 23, corresponde a 0,92 (92%) e se errou 2 músicas corresponde a 0,08 (8%). Com isto é possível concluir que o modelo tem mais facilidade em classificar as músicas alegres, porque teve 23 acertos em 25, e tem mais dificuldade em classificar músicas calmas, porque teve apenas 5 acertos em 16.

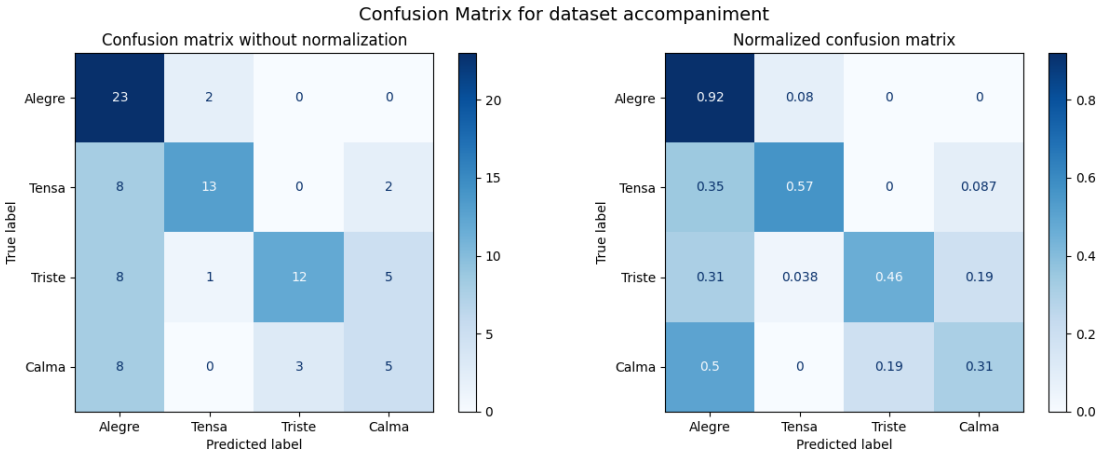


Figura 41 - Matriz de confusão obtida para o classificador de acompanhamento (90% treino, 10% teste)

Capítulo 4
Planeamento da aplicação final

O desenvolvimento da solução seguiu uma metodologia ágil[71] com diversas iterações ao longo das várias fases do projeto. No início de cada iteração havia uma reunião com o orientador para planear as tarefas a executar na iteração atual. Todos os ciclos tiveram uma data-limite que não era muito rigorosa, porque existiram um conjunto de variantes que nem sempre permitiam terminar as tarefas até à data estipulada, mas assim que terminadas as tarefas eram apresentadas, revistas, discutidas as soluções propostas para os problemas encontrados, e planeada a próxima iteração. Foi utilizada a ferramenta Trello[72] para facilitar a gestão de tarefas, associando as mesmas com os estados por realizar, a realizar e realizadas. Ao longo do projeto, não houve muito rigor na utilização desta ferramenta, servindo mais para colocar as linhas gerais e tarefas de alto nível a resolver. O Slack[73], uma plataforma de comunicação que permite integrar outras aplicações, foi uma ferramenta chave para a equipa, pois permitiu manter um canal de comunicação com a equipa (orientador e João Canoso), que ajudou a desbloquear problemas, trocando informações quando necessário e na motivação pelo progresso continuo de trabalho, criando hábitos e rotinas.

A solução proposta consiste numa solução bi-modal de reconhecimento de emoções em música utilizando as componentes de áudio e lírica. Este sistema continua a seguir uma arquitetura de microsserviços utilizando filas de mensagens para comunicação, como ilustrado de exemplo na Figura 42, como feito anteriormente em (António, 2019). Optou-se por continuar a utilizar este tipo de arquitetura pelas razões apresentadas nesse mesmo trabalho, fundamentalmente "dada a oferta crescente de soluções de computação para a nuvem e considerando as especificidades deste projeto, torna-se muito mais acertada a utilização de uma arquitetura de microsserviços, quer em termos de custos, quer em termos de gestão, manutenção e desenvolvimento contínuo da aplicação" (António, 2019).

[71] https://en.wikipedia.org/wiki/Agile_software_development
[72] https://trello.com/
[73] https://slack.com/intl/en-pt/

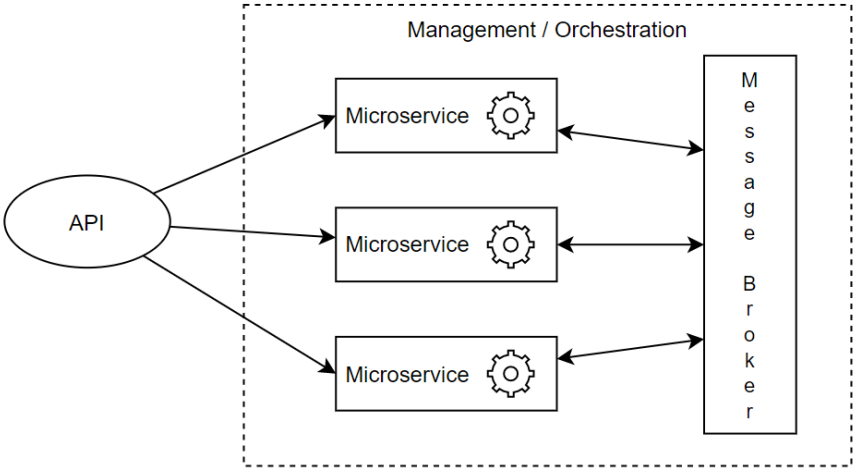


Figura 42 - Exemplo de arquitetura de microsserviços que utiliza filas de mensagens[74]

A gestão e orquestração dos microsserviços não é contemplada neste trabalho, é estudada e implementada num trabalho paralelo realizado pelo aluno João Canoso, também do Instituto Politécnico de Tomar. Assim, será apenas descrito na secção 4.1 no que consiste este tipo de arquitetura e a tecnologia escolhida para a implementar. Na secção 4.2 é também descrita a lógica de comunicação entre os processos utilizando o serviço de filas de mensagens RabbitMQ. Esta solução foi alvo de estudo no protótipo inicial, sendo por isso justificada em (António, 2019). Para terminar, são apresentadas de forma breve as linguagens de programação utilizadas para desenvolver o sistema, na secção 4.3, e na secção 4.4 a biblioteca FFMPEG que é utilizada em diversos momentos para a manipulação do áudio.

4.1. Arquitetura de microsserviços

Microsserviços são uma das tendências que mais cresceu nos últimos anos para o desenvolvimento de aplicações complexas. (Ghofrani & Lübke, 2018) Uma arquitetura de microsserviços consiste num conjunto de pequenos serviços isolados e autónomos que podem ser executados independentemente uns dos outros, mas que trabalham juntos para atingir uma finalidade.

[74] Baseada na figura de https://feras.blog/microservices-architecture-to-be-or-not-to-be/

Os microsserviços são desenvolvidos e mantidos separadamente, permitindo que cada um deles possa utilizar tecnologias diferentes se necessário, como por exemplo um ser desenvolvido numa linguagem de programação diferente de outro. "Os benefícios mais importantes do uso de microsserviços são agilidade, autonomia, escalabilidade, resiliência e fácil implantação (*deploy*) contínua." (Vural et al., 2017)

Como descrito anteriormente, a componente de gerir e orquestrar os microsserviços não consta neste trabalho, já que é foco de um trabalho de mestrado a ser realizado em paralelo. Nesse trabalho, após avaliar as diversas possibilidades, foi decidido que a tecnologia a utilizar para gerir e orquestrar os microsserviços seria Kubernetes.

Kubernetes, também conhecido como k8s, é um sistema de código-aberto para *deploy* automatizado, escalável e permite gerir aplicações que se baseiam em *containers*, cuja arquitetura é ilustrada na Figura 43. Por norma é uma abordagem bastante flexível, permitindo à aplicação crescer consoante as necessidades, isto é, iniciar ou terminar serviços/*containers* paralelos sempre que a carga de trabalho o justifique, escalando sempre que necessário. Fornece ainda um conjunto de outras funcionalidades vantajosas, entre elas orquestração de armazenamento, *self-healing* (por exemplo reinicia os *containers* que falham) e gestão de segredos e configurações privadas.[75]

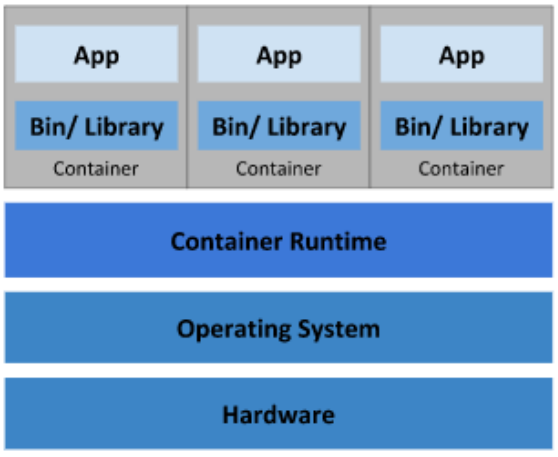


Figura 43 – Estrutura de 3 aplicações *containerizadas* no mesmo sistema

[75] https://kubernetes.io/

4.2. Comunicação entre microsserviços

Sendo uma arquitetura de microsserviços um conjunto de pequenos serviços isolados, estes necessitam de comunicar entre si de algum modo. A comunicação entre estes é feita utilizando o mecanismo de filas de mensagens RabbitMQ que, entre várias alternativas, foi escolhido após o estudo feito durante a prova de conceito (António, 2019).

Este tipo de mecanismos fornece comunicação assíncrona, permitindo que os microsserviços comuniquem entre si através de um intermediário (*broker*) sem estarem acoplados. A função do intermediário é traduzir uma mensagem do protocolo de mensagens do emissor para o protocolo de mensagens do recetor[76]. Isto permite ter centralizado o armazenamento e a gestão dos dados, para não existir perda de dados em caso de falha do sistema porque as mensagens ficam nas filas até serem consumidas.

Existem dois modelos de operação para estes mecanismos: o modo de *queueing* (produtor/consumidor) e *publish*/*subscribe* (publicar/subscrever). Tendo por base a arquitetura de microsserviços, no primeiro modo os microsserviços produzem mensagens para as filas, onde depois cada mensagem é consumida por um único microsserviço. No segundo, os microsserviços publicam mensagens para as filas e estas são entregues a todos os microsserviços que estejam à escuta (subscrever) daquelas filas. Ambos os modos têm vantagens e desvantagens, resolvendo problemas distintos. Para o nosso sistema foi escolhido o modo de *queueing* porque as mensagens são consumidas apenas por um único consumidor. Esta opção permite que possa existir mais do que um microsserviço a consumir a mesma fila de mensagens sem haver duplicação de trabalho, ou seja, imaginando que existem dois microsserviços para descarregar vídeos a consumir uma fila, ao ser inserida uma mensagem nessa fila apenas um dos microsserviços consome a mensagem, iniciando a tarefa nela descrita. Ao ser introduzida outra mensagem, será consumida pelo outro, sendo impossível que ambos os microsserviços efetuem a mesma tarefa (descarregar o mesmo vídeo).

[76] https://en.wikipedia.org/wiki/Message_broker

RabbitMQ é um *software* de código aberto que neste sistema fará o papel de intermediário de mensagens, aceitando e encaminhando as mesmas. O principal papel deste intermediário de mensagens é "receber mensagens dos *publishers* (aplicações que as publicam, também conhecidos como produtores) e encaminham-nas para os consumidores (aplicações que as processam)".[77] As mensagens são armazenadas no intermediário, ficando seguras até serem consumidas. É leve e fácil de implantar (fazer *deploy*) até em ambiente distribuído para obter alta disponibilidade e rendimento tanto localmente como em *cloud* (nuvem)[78]. As mensagens a enviar e receber são separadas visto que se trata de um mecanismo assíncrono. Pode ser integrado utilizando diversas linguagens de programação[79] e disponibiliza diversas funcionalidades, entre elas a gestão e monitorização das filas de mensagens.

RabbitMQ suporta diversos protocolos, mas o protocolo utilizado no modo de *queueing* é o AMQP 0-9-1 que permite que aplicações clientes comuniquem com intermediários de mensagens em conformidade. Uma visão geral sobre este protocolo está ilustrada na Figura 44 onde as mensagens são publicadas para um mecanismo presente no intermediário de mensagens chamado *Exchange*. Este mecanismo tem a função de receber as mensagens dos produtores e distribuir as mesmas pelas filas respetivas. Quando as mensagens se encontram nas filas podem ser consumidas pelos consumidores, desaparecendo da fila.

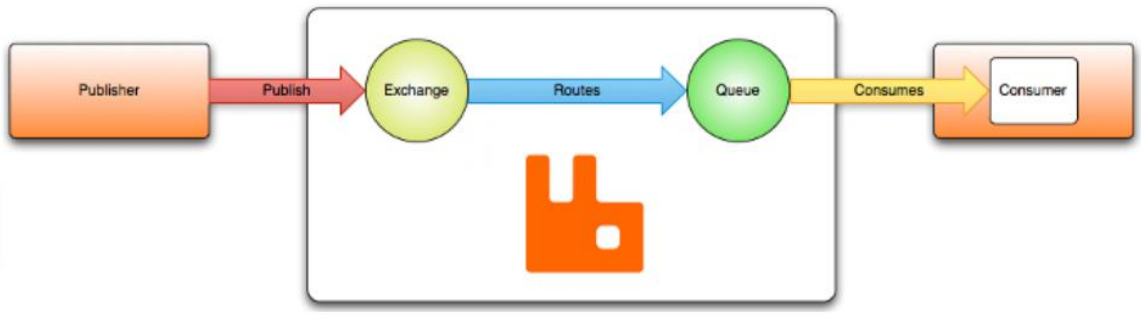


Figura 44 - Visão geral do protocolo AMQP 0-9-1[80]

[77] https://www.rabbitmq.com/tutorials/amqp-concepts.html#brokers-role
[78] https://www.rabbitmq.com/#features
[79] https://www.rabbitmq.com/getstarted.html
[80] https://www.rabbitmq.com/tutorials/amqp-concepts.html#amqp-model

4.3. Linguagens de programação utilizadas

Existem muitas linguagens de programação disponíveis, sendo que a maioria pode ser utilizada para diversas funcionalidades, mas existe uma ou várias áreas onde cada uma se enquadra melhor. Por exemplo, a linguagem Python pode ser utilizada para o desenvolvimento *web* ou de aplicações móveis, mas atualmente um dos seus pontos fortes é na área de *data science* (ciência dos dados). Sendo que a arquitetura do sistema é a de microsserviços, não existe a necessidade de restringir o desenvolvimento a uma única linguagem de programação, podem ser utilizadas várias e, no limite, cada microsserviço até pode ter uma linguagem de programação diferente. As linguagens de programação escolhidas foram Node.js e Python, uma vez que de acordo com a análise feita foram as que melhor supriam as necessidades para o projeto e o conhecimento da equipa envolvida. Sendo a arquitetura de microsserviços, faria todo o sentido utilizar uma linguagem escalável como Node.js, com um ecossistema rico (bibliotecas) e com bom desempenho. Por outro lado, considerando as tecnologias a utilizar no estado de arte, muitas delas fornecendo bibliotecas para Python, faria todo o sentido considerar esta linguagem para as tarefas de inteligência artificial. Estas opções haviam já sido seguidas no trabalho anterior (António, 2019), sendo a análise e justificação apresentada nesse trabalho bastante válida e por isso contribuindo também para esta escolha.

4.4. Manipulação do áudio

Como exposto na secção 1.2.2 onde é descrita a solução que se pretende obter, a manipulação do áudio é algo crucial para o desenvolvimento deste trabalho. Isto porque pretende-se seguir a literatura do reconhecimento de emoções em música e, por isso, os ficheiros de áudio são segmentados em excertos com tamanho fixo. Para além desse motivo, as músicas a classificar são provenientes do YouTube, sendo que um microsserviço descarrega o vídeo com o formato mp4[81], carregando as componentes de áudio e vídeo. Como o vídeo não interessa para a extração de caraterísticas áudio, o ficheiro deve ser convertido para um

[81] https://en.wikipedia.org/wiki/MPEG-4_Part_14

formato de áudio, sendo os mais habituais mp3[82] (MPEG-1/2 Audio Layer III) e wav[83] (Waveform). Entre ambos, optou-se por converter os ficheiros para o formato wav porque, embora seja mais pesado em termos de armazenamento, acaba por ser mais leve em termos de processamento. Por outro lado, durante o processo de treino dos classificadores, os ficheiros do *dataset* de áudio estão no formato mp3 e por isso devem também ser convertidos para wav. Por último, é ainda aplicado o processo de *downsampling* para diminuir o número de amostras nos áudios, a fim de diminuir as necessidades computacionais.

As afirmações acima justificam o uso de uma tecnologia que faça a manipulação do áudio e a tecnologia escolhida foi FFMPEG. É um *software* de código aberto, grátis e multiplataforma que consiste num vasto conjunto de bibliotecas e programas para trabalhar com vídeo, áudio e outros ficheiros de multimédia. O núcleo é o próprio programa FFMPEG, projetado para processamento de ficheiros de vídeo e áudio através da linha de comandos. As várias bibliotecas que este contém permitem entre várias funcionalidades, aquelas que são necessárias a este trabalho, portanto rotinas para alterar a taxa de amostragem, converter os ficheiros entre diversos formatos e segmentação.[84]

[82] https://en.wikipedia.org/wiki/MP3
[83] https://en.wikipedia.org/wiki/WAV
[84] https://en.wikipedia.org/wiki/FFmpeg

Capítulo 5
Desenvolvimento da aplicação final

O desenvolvimento da solução seguiu uma metodologia ágil[85] com diversas iterações ao longo das várias fases do projeto. No início de cada iteração havia uma reunião com o orientador para planear as tarefas a executar na iteração atual. Todos os ciclos tiveram uma data-limite que não era muito rigorosa, porque existiram um conjunto de variantes que nem sempre permitiam terminar as tarefas até à data estipulada, mas assim que terminadas as tarefas eram apresentadas, revistas, discutidas as soluções propostas para os problemas encontrados, e planeada a próxima iteração. Foi utilizada a ferramenta Trello[86] para facilitar a gestão de tarefas, associando as mesmas com os estados por realizar, a realizar e realizadas. Ao longo do projeto, não houve muito rigor na utilização desta ferramenta, servindo mais para colocar as linhas gerais e tarefas de alto nível a resolver. O Slack[87], uma plataforma de comunicação que permite integrar outras aplicações, foi uma ferramenta chave para a equipa, pois permitiu manter um canal de comunicação com a equipa (orientador e João Canoso), que ajudou a desbloquear problemas, trocando informações quando necessário e na motivação pelo progresso continuo de trabalho, criando hábitos e rotinas.

A fase inicial consistiu em ler, compreender e colocar em funcionamento a solução anterior. As várias tecnologias que foram utilizadas são atuais e as respetivas decisões pelos seus usos são compreensíveis. O sistema em termos gerais é simples, mas eficiente, permitindo atingir os resultados a que se propunha. Possivelmente por questões de tempo, não foi possível estudar algumas tecnologias utilizadas tão profundamente e, portanto, existia desde logo algumas melhorias e lacunas que eram necessárias fazer quanto à sua implementação, e até mesmo para colocar o sistema em funcionamento, como por exemplo eliminar a utilização de endereços IP estáticos nos microsserviços porque a sua utilização não é de todo recomendada em produção.

[85] https://en.wikipedia.org/wiki/Agile_software_development
[86] https://trello.com/
[87] https://slack.com/intl/en-pt/

Na fase seguinte foram realizados diferentes testes com base no estudo das várias tecnologias descritas no capítulo 2. Esta fase permitiu avaliar a viabilidade das diversas tecnologias, antecipando um possível risco de falha num período mais avançado do desenvolvimento. Após avaliadas e escolhidas as tecnologias a utilizar, foi iniciado o desenvolvimento das várias tarefas isoladas do sistema utilizando as respetivas tecnologias para atingir os determinados fins. Como se pretendia que o tipo de arquitetura a utilizar fosse uma Arquitetura de Microsserviços implementados sob forma de *containers*, o ambiente de desenvolvimento teria de se parecer o mais possível ao ambiente de produção, portanto como simulação dos *containers* foi utilizada a tecnologia Vagrant[88] para desenvolver e alojar os microsserviços em Docker, visto já existir experiência em utilizar o mesmo. Vagrant é uma ferramenta focada na portabilidade e automatização para construir e gerir ambientes isolados de máquinas virtuais, neste caso o *software* de virtualização utilizado foi VirtualBox[89]. Com isto, foram criadas diversas máquinas virtuais para simular cada um microsserviços, que ficam assim isolados e executam o seu código independente.

A arquitetura do sistema está ilustrada na Figura 45 e cada microsserviço é descrito pelas seguintes secções. A Tabela 7 resume o mapeamento entre os microsserviços e as filas de mensagens para que cada um produz e consome, mas é importante notar ao analisar essa figura que cada microsserviço consome uma fila de mensagens exclusiva, à exceção do Manager que produz para todas as outras filas, mas todos eles produzem mensagens para a fila *Management*, que é a fila onde o microsserviço Manager consome mensagens porque é este que dita a sequência de instruções, o passo a passo do sistema.

[88] https://www.vagrantup.com/
[89] https://www.virtualbox.org/

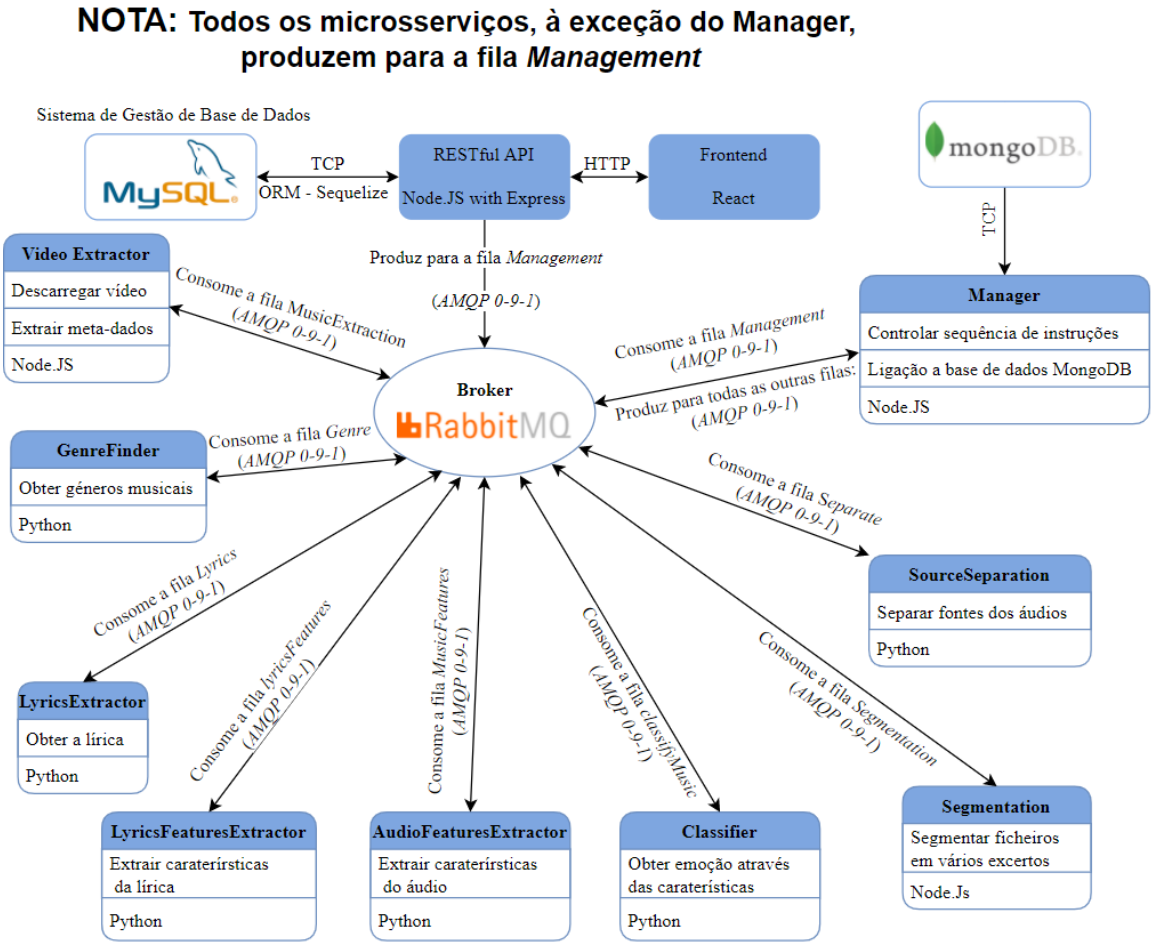
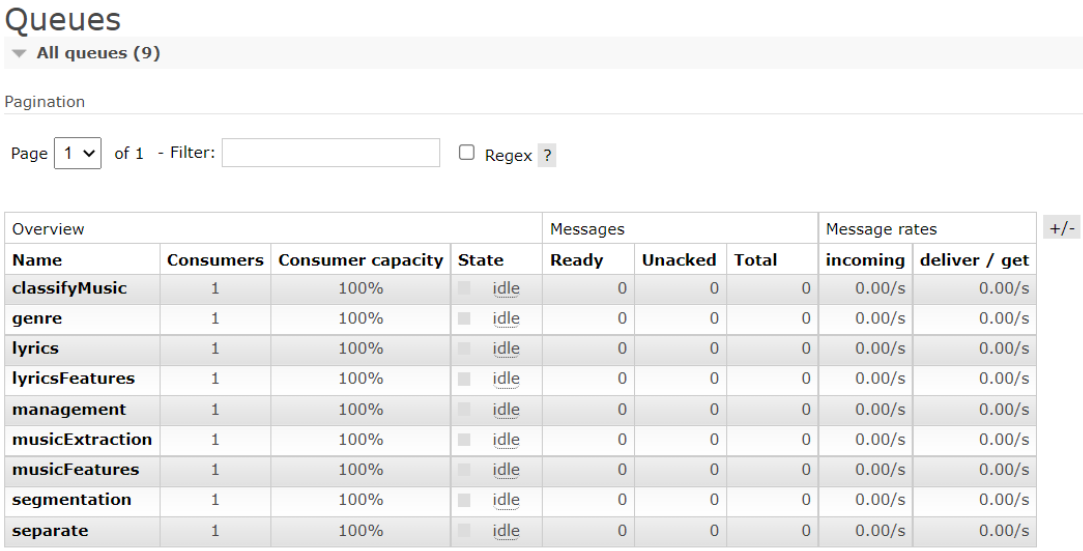


Figura 45 - Arquitetura geral da comunicação entre microsserviços na solução implementada

5.1. Comunicação entre microsserviços

O intermediário de mensagens (broker) escolhido foi RabbitMQ, como descrito na secção 4.2, e este tem por objetivo gerir as comunicações entre todos os microsserviços. Todos os microsserviços estão ligados permanentemente ao intermediário de mensagens através de canais específicos. Existindo estas ligações, cada um deles consome e produz mensagens para alguma das filas que foram atualmente implementadas e estão ilustradas na Figura 46 através da página *web* disponibilizada pelo RabbitMQ para monitorização do intermediador de mensagens.



Queues

All queues (9)

Pagination

Page 1 of 1 - Filter: Regex ?

Overview				Messages			Message rates		+/-
Name	Consumers	Consumer capacity	State	Ready	Unacked	Total	incoming	deliver / get	
classifyMusic	1	100%	idle	0	0	0	0.00/s	0.00/s	
genre	1	100%	idle	0	0	0	0.00/s	0.00/s	
lyrics	1	100%	idle	0	0	0	0.00/s	0.00/s	
lyricsFeatures	1	100%	idle	0	0	0	0.00/s	0.00/s	
management	1	100%	idle	0	0	0	0.00/s	0.00/s	
musicExtraction	1	100%	idle	0	0	0	0.00/s	0.00/s	
musicFeatures	1	100%	idle	0	0	0	0.00/s	0.00/s	
segmentation	1	100%	idle	0	0	0	0.00/s	0.00/s	
separate	1	100%	idle	0	0	0	0.00/s	0.00/s	

Figura 46 - Filas de mensagens implementadas

O microsserviço Manager determina a sequência de instruções do sistema. Por essa razão, todos os microsserviços produzem as mensagens com o resultado do seu trabalho para a fila de mensagens que este está a consumir, fila *Management*. Dependendo das mensagens que aí recebe, este serviço Manager irá determinar a próxima tarefa a executar pelo sistema, colocando novas mensagens com esta instrução (ou instruções) nas várias filas específicas. A Tabela 7 mostra o mapeamento entre os microsserviços e as filas de mensagens para os quais produzem e consomem.

	Filas de Mensagens	
Microsserviços	**Consome de**	**Produz para**
Frontend	NA	NA
Backend	NA	Management
Manager	Management	Todas as outras filas
GenreFinder	Genre	Management
LyricsExtractor	Lyrics	Management
LyricsFeaturesExtractor	lyricsFeatures	Management
VidExtractor	MusicExtraction	Management

SourceSeparation	Separate	Management
Segmentation	Segmentation	Management
AudioFeaturesExtractor	MusicFeatures	Management
Classifier	classifyMusic	Management

Tabela 7 - Tabela de mapeamento entre microsserviços e filas de mensagens

Todas as mensagens seguem a estrutura que está ilustrada no exemplo da Figura 47, onde o campo *Service* corresponde ao nome do microsserviço que produz a mensagem e o campo *Result* contém o resultado do microsserviço sob a forma de objeto, podendo este ter diversos campos, por exemplo o identificador do vídeo, a lírica da música, o género musical, caraterísticas extraídas, etc.

```
{
    Service: "<microservice name>",
    Result: {
        Field_1: "<value>",
        Field_2: "<value>",
        Field_N: "<value>"
    }
}
```

Figura 47 - Exemplo de estrutura das mensagens

5.2. Aplicação *web*

Na prova de conceito inicial foi desenvolvida uma aplicação web que, entre diferentes funcionalidades, permitia ao utilizador pedir para classificar emocionalmente uma música através do seu endereço URL do YouTube e consultar as músicas que já tinham sido classificadas, apresentando alguns detalhes retirados do YouTube em tempo real, como o número de visualizações das músicas. A aplicação estava pronta para apresentar ao utilizador uma única emoção associada a cada música, pelo que foi necessário fazer alterações à mesma.

Como mostra a Figura 45, a aplicação *web* comunica com uma base de dados SQL. O modelo de dados teve de sofrer modificações porque existia apenas uma tabela para guardar as informações básicas das músicas a apresentar ao utilizador, onde se inseria a emoção. Como agora existem diversas emoções associadas a cada música, tomou-se a opção de criar uma nova tabela só para guardar as emoções, ficando ligada à tabela que guarda as informações das músicas.

Também foi necessário alterar a API em diversos tópicos, mas a componente mais importante foi desenvolver um raciocínio que permitisse à API, ao receber do microsserviço Manager as várias emoções de uma música, traduzir aquele resultado e introduzir corretamente na base de dados SQL para apresentar ao utilizador.

A componente de *frontend* sofreu algumas mudanças de estética e foram implementadas e removidas várias funcionalidades. Algumas das novas funcionalidades são a possibilidade de visualizar a forma de onda (*waveform*) de cada música, assim como as várias emoções associadas às diferentes fontes musicais e excertos.

5.3. Manager

O microsserviço Manager tem por objetivo gerir todos os outros microsserviços, determinando a sequência de instruções do sistema. Este foi desenvolvido para tornar os microsserviços mais independentes entre eles porque anteriormente como os microsserviços se chamavam uns aos outros, existia sempre um grau de dependência entre eles, mas a utilização deste microsserviço permite que os microsserviços não estejam dependentes uns com os outros, apenas com o manager e, na necessidade de alterar a sequência de instruções, basta alterar no Manager. Este foi desenvolvido utilizando Node.js e consome a fila de mensagens *Management* ligando-se ao intermediador de mensagens através da biblioteca amqplib[90]. Esta é a biblioteca utilizada também na página oficial do RabbitMQ para exemplificar as suas funcionalidades, permitindo criar clientes que utilizam o protocolo

[90] https://www.npmjs.com/package/amqplib

AMQP 0-9-1 e assim aproveitar o modo *queueing* para então ter os microsserviços a consumir e produzir mensagens.

É apenas neste microsserviço que existe ligação à base de dados NoSQL. A base de dados NoSQL escolhida foi MongoDB[91] principalmente pela facilidade em aprender e utilizar a mesma, pela sua popularidade e pelo facto deste modelo de dados suportar JSON, onde já existe experiência. A biblioteca MongoDB[92] é o motor oficial do MongoDB para Node.js e por isso foi naturalmente escolhida, possibilitando fazer todos os comandos necessários para criar, visualizar, atualizar e eliminar documentos. Cada música tem um documento individual na base de dados, onde as várias informações vão sendo preenchidas à medida que são obtidas nos respetivos microsserviços, estando representado na Figura 48 um exemplar da estrutura dos documentos, onde o número de caraterísticas musicais foi reduzido por questões de espaço na página. Na sua estrutura, o *videoID* corresponde ao identificador do vídeo, *song* e *artist* correspondem ao título da música e nome do artista respetivamente, *accompaniment*, *original* e *vocal* são listas que contêm as caraterísticas extraídas dos ficheiros de áudio de cada fonte previamente separada, que vão sendo inseridas à medida que vão ficando disponíveis. A posição destas caraterísticas nas listas é de extrema importância pois permite saber a que excerto correspondem as mesmas, por exemplo as caraterísticas do primeiro excerto estão na primeira posição da lista, as caraterísticas do segundo excerto estão na segunda posição da lista e assim por diante. As entradas *emotions_accompaniment*, *emotions_original*, *emotions_vocals*, *emotions_allaudio* e *emotions_lyrics* contêm as emoções atribuídas pelos classificadores para cada excerto de cada fonte, sendo igualmente listas e, assim como explicado acima, a posição importa para saber a que excerto corresponde cada emoção. Exceção a isto é a lírica, já que não é segmentada e por isso apenas se obtém uma emoção única. *Genre* contém os cinco géneros musicais mais populares da música, *lyrics* contém a lírica obtida, *lyricsFeatures* são as caraterísticas obtidas da lírica e *numFiles* é a quantidade de excertos em que o áudio foi segmentado, que ajudará depois na parte de classificação.

[91] https://www.mongodb.com/
[92] https://www.npmjs.com/package/mongodb

```
{
    "videoID":"ALZHF5UqnU4",
    "song":"Alone",
    "artist":"Marshmello",
    "accompaniment":[
        { "lowlevel_average_loudness_accompaniment":0.6595616340637207,
            "lowlevel_barkbands_crest_dmean_accompaniment":2.6492116451263428 },
        { "lowlevel_average_loudness_accompaniment":0.9534558057785034,
            "lowlevel_barkbands_crest_dmean_accompaniment":8.19078254699707 }
    ],
    "original":[
        { "lowlevel_average_loudness_original":0.6148363947868347,
          "lowlevel_barkbands_crest_dmean_original":2.8198087215423584 },
        { "lowlevel_average_loudness_original":0.9544604420661926,
          "lowlevel_barkbands_crest_dmean_original":2.745400905609131 }
    ],
    "vocals":[
        { "lowlevel_average_loudness_vocals":0.07123716920614243,
          "lowlevel_barkbands_crest_dmean_vocals":3.0628721714019775 },
        { "lowlevel_average_loudness_vocals":0.7735997438430786,
          "lowlevel_barkbands_crest_dmean_vocals":3.8258256912231445 }
    ],
    "emotions_accompaniment":[ "Alegre", "Calma" ],
    "emotions_original":[ "Triste", "Tensa" ],
    "emotions_vocals":[ "Alegre", "Alegre" ],
    "emotions_allaudio":[ "Triste", "Alegre" ],
    "emotions_lyrics":[ "Triste" ],
    "genre":"electronic, electro, alone, amazing, sweet",
    "lyrics":"\"Alone\" by Marshmello:\n    I'm so alone, nothing feels like home...",
    "lyricsFeatures":{ "Semantic_Gazeteers_AvgValence":0, "Semantic_Gazeteers_AvgArousal":0,
        "Semantic_DAL_ANEW_AvgValence":" 1.9989", "Semantic_DAL_ANEW_AvgArousal":" 1.581229" },
    "numFiles":2
}
```


Figura 48 - Estrutura dos documentos da base de dados NoSQL

Inicialmente o microsserviço cria uma ligação ao modelo de dados MongoDB, selecionando a base de dados e a coleção (equivale a uma tabela em SQL) a utilizar. De seguida aguarda por novas mensagens na fila *Management*, que seguem a estrutura da Figura 47. Quando recebe uma mensagem, o que define as ações a executar é com base no campo *Service* da mensagem porque o microsserviço foi desenvolvido com uma estrutura de decisão, um *switch*, que irá fazer determinadas ações consoante o valor desse *Service*.

O campo *service* contém o nome do microsserviço que produziu as mensagens, ou seja, se por exemplo o Manager receber uma mensagem a dizer API no campo *service*, então significa que recebeu uma mensagem do microsserviço API. No caso de receber uma mensagem do microsserviço:

- API – irá enviar o endereço URL (Uniform Resource Locator) para o Microsserviço VidExtractor descarregar o vídeo e alguns meta-dados;
- VidExtractor – se receber a informação de que o vídeo não pertence à categoria música, termina o processo de imediato. Senão irá criar um documento na base de dados NoSQL para aquele vídeo, com as informações recebidas do identificador do vídeo, título da música e nome do artista. Irá enviar estas informações para os microsserviços LyricsExtractor e GenreFinder, e irá enviar o apenas identificador do vídeo para o SourceSeparation;
- GenreFinder – atualiza o documento na base de dados com os géneros recebidos ou mensagem de erro;
- SourceSeparation – irá enviar para o microsserviço Segmentation o identificador do vídeo para segmentar os ficheiros;
- Segmentation – atualiza na base de dados o campo que contém o número de excertos em que foi segmentado o vídeo. De seguida, para cada ficheiro novo originado na segmentação, será enviada a sua identificação para o microsserviço AudioFeaturesExtractor extrair as características;
- AudioFeaturesExtractor – adiciona à respetiva lista de caraterísticas, dependendo da fonte musical, no documento da base de dados com as características extraídas ou mensagem de erro. No caso de serem recebidas as caraterísticas do último ficheiro daquele vídeo, então está pronto para classificar o vídeo, portanto para cada excerto será enviado para o microsserviço Classifier quatro mensagens, cada uma com as características de uma fonte, ou seja, será enviado para cada excerto uma mensagem com as características do acompanhamento, outro com o original, com a componente vocal e outra mensagem com todas as características juntas.
- LyricsExtractor – no caso de ter sido impossível obter a lírica, será atualizado na base de dados com essa informação no campo *lyrics* e a parte de lírica termina aqui. Senão, será adicionada a lírica na base de dados e enviado o nome do ficheiro que a contém para o microsserviço LyricsFeaturesExtractor extrair caraterísticas da mesma;
- LyricsFeaturesExtractor – São guardadas as caraterísticas na base de dados e enviadas para o microsserviço Classifier classificar.

- Classifier – adiciona à base de dados NoSQL, no respetivo campo da fonte musical, a emoção classificada. Quando todos os excertos de um vídeo forem classificados, é enviado para a API todas as emoções classificadas de um vídeo para adicionar na base de dados SQL e poderem ser apresentadas aos utilizadores.

5.4. VidExtractor

Este microsserviço foi desenvolvido para descarregar o vídeo que se pretende classificar emocionalmente e extrair alguns dos seus meta-dados, nomeadamente título da música e nome do artista. Consome a fila de mensagens *MusicExtraction*, produz para a fila *Management* e a lógica implementada é descrita de seguida e está ilustrada na Figura 49.

No sistema anterior (António, 2019) já existia um microsserviço desenvolvido em Node.js que descarregava vídeos. Este estava ligado ao intermediador de mensagens utilizando a biblioteca amqplib e recebia o identificador do vídeo por mensagem. De seguida, utilizando a biblioteca ytdl-core[93], validava se o vídeo pertencia à categoria de música e em caso negativo terminava de imediato. Em caso positivo descarregava o vídeo no formato mp4 que era posteriormente guardado com a extensão wav para ser um ficheiro de áudio, mas continuava a ter o formato mp4 visto que não existia realmente uma conversão, apenas era alterada a extensão e por isso incluía as duas componentes de áudio e vídeo.

Tendo em conta que se pretende descarregar o vídeo na mesma, foi realizada uma análise sobre o código implementado, pelo que se manteve praticamente tudo, exceto a maneira como o vídeo era guardado pois no momento só se pretende ter realmente a componente de áudio, portanto não existe necessidade de manter a componente de vídeo. Logo, em vez de apenas alterar a extensão do ficheiro de vídeo para wav para ser identificado como um ficheiro de áudio, é realizado um processo de conversão utilizando a biblioteca fluent-ffmpeg[94], como já havia sido referido na subsecção 4.4 existem vários momentos onde são realizados procedimentos de manipulação do áudio e foi decidido utilizar o FFMPEG. Os

[93] https://www.npmjs.com/package/ytdl-core
[94] https://www.npmjs.com/package/fluent-ffmpeg

vídeos são todos descarregados para a mesma diretoria, sendo os seus nomes formados pelos identificadores dos vídeos com a estrutura "<videoID>.wav". A diretoria onde estes se encontram é uma diretoria partilhada entre todos os microsserviços.

Para além da tarefa de descarregar o vídeo, são também obtidos o nome da música e do artista utilizando mais uma vez a biblioteca ytdl-core, a fim de conseguir obter o género musical e a lírica da música mais adiante. Mas nem todos os vídeos têm estes meta-dados definidos manualmente, principalmente os vídeos mais antigos, pelo que na impossibilidade de os obter segundo esse método, está a ser realizada outra abordagem para os obter através do título do vídeo, pois habitualmente os vídeos do YouTube de músicas têm um formato específico que é <Nome do artista> - <Nome da música>. Portanto está a ser verificado se o título contém o caracter – (travessão) e, ao confirmar-se, está a ser considerado, sem qualquer validação, que a primeira parte é o nome do artista e a segunda parte o nome da música.

Por fim é produzida uma mensagem apenas com o identificador do vídeo na impossibilidade de obter os meta-dados, visto que o vídeo foi descarregado na mesma e alguns dos microsserviços seguintes necessitam de saber sobre que ficheiro vão trabalhar, ou então é produzia uma mensagem com o identificador do vídeo e os respetivos meta-dados para que o Manager crie um documento na base de dados NoSQL com estas novas informações para cada áudio com a estrutura da Figura 48.

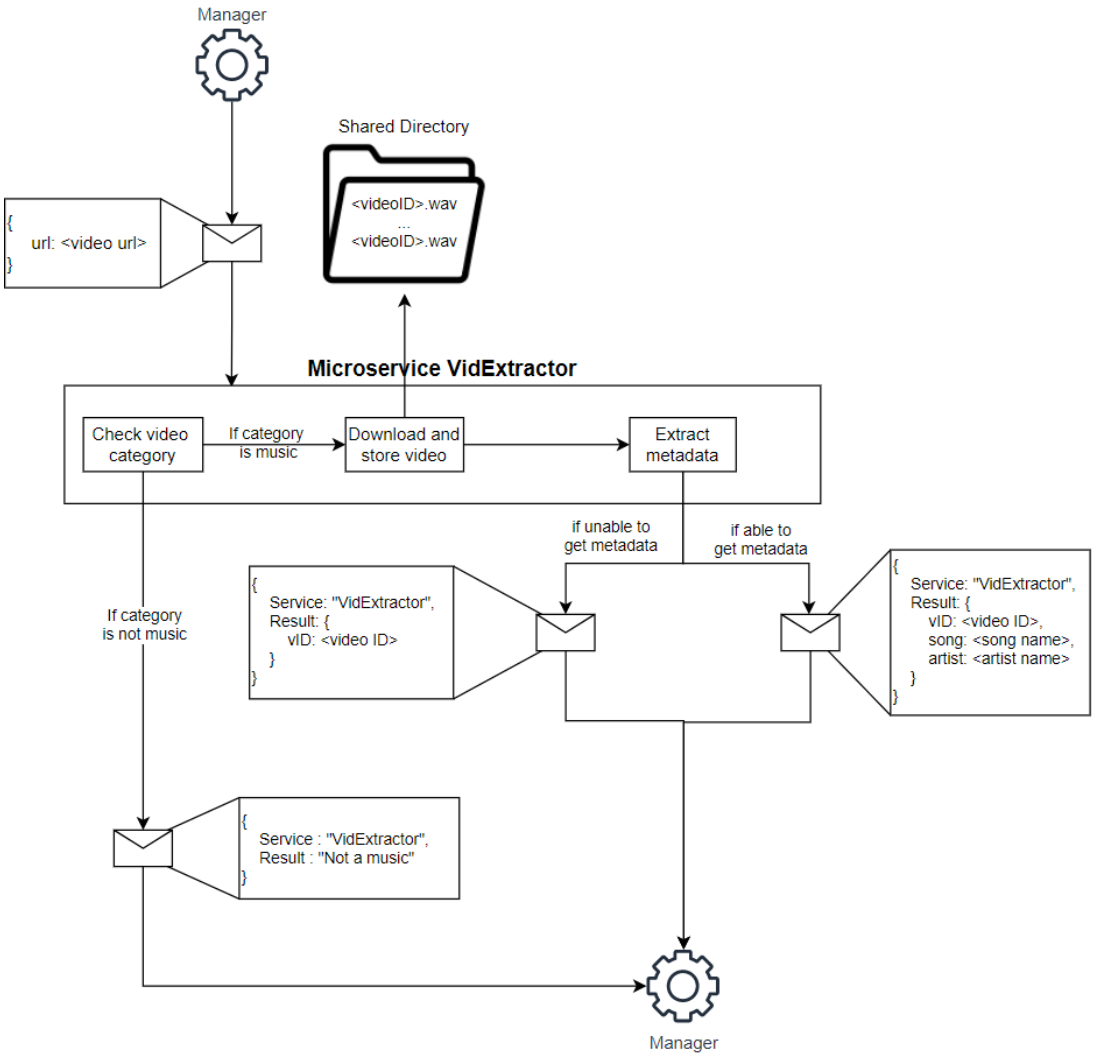


Figura 49 - Lógica do microsserviço vidExtractor

5.5. GenreFinder

O objetivo deste microsserviço é obter os géneros musicais dos áudios. A intenção inicial para o desenvolvimento do mesmo, era para utilizar o género musical como caraterística, juntamente com as restantes caraterísticas extraídas do áudio para a classificação. Isto porque o *dataset* de áudios incluía alguns meta-dados extraídos, entre eles a lista de géneros musicais para cada áudio.

Sendo que os ambientes de treino e real devem ser o mais idênticos possível, a nomenclatura dos géneros musicais convém ser a mesma para em ambos os ambientes, para não estar a ser

treinado um classificador com uns géneros musicais e depois aparecerem outros diferentes no ambiente real. Logo, era essencial que a API que foi utilizada para extrair os géneros musicais do *dataset* fosse a mesma que seria utilizada no ambiente real. Na altura tinha sido utilizada a API da Rovi[95] e por isso foi feito um registo para obter uma chave de acesso. No entanto, após esperar um tempo considerável e três tentativas de registo, não foi obtida qualquer resposta, restando duas opções: extrair o género musical para todos os áudios do *dataset* utilizando outra API ou simplesmente descartar a possibilidade do seu uso no procedimento de classificação. Por questões de tempo, tornou-se impraticável a primeira opção, mas não foi totalmente descartada a utilização deste microsserviço. Seguiu-se a ideia de desenvolver na mesma o microsserviço para obter o género, que por enquanto será apenas utilizado para efeitos demonstrativos, associando-se aos restantes meta-dados disponíveis na aplicação *web*, num trabalho futuro de integração na classificação.

O microsserviço foi desenvolvido em Python e consome a fila de mensagens *Genre* através da biblioteca Pika[96]. Esta biblioteca é também ela utilizada na página oficial do RabbitMQ para demonstrações, sendo uma implementação em Python do protocolo AMQP 0-9-1.

As mensagens consumidas contêm três parâmetros: o identificador do vídeo, o título da música e o nome do artista caso tenha sido possível de obter. Está a ser utilizada a API da Last.FM[97] que é bastante rica, disponibilizando diversos métodos para obter variados meta-dados, entre eles uma lista de géneros musicais para cada áudio. Para tal necessita de dois parâmetros, o título da música e o nome do artista que recebe através da mensagem. Como demonstrado na Figura 50, é recebida então uma mensagem com o identificador do vídeo e pode ou não conter o nome do artista e título da música, que no caso de não conter, pela impossibilidade de ser conseguida previamente, torna-se incapaz de obter o género musical e, portanto, será produzida uma mensagem que segue a mesma estrutura das outras, que está ilustrada na Figura 50 pelo ramo “Unable to get Genre” e colocada na fila de mensagens *Management*. No caso da mensagem conter o título e nome do artista irá utilizar a API para

[95] http://developer.rovicorp.com/docs
[96] https://pika.readthedocs.io/en/stable/
[97] https://www.last.fm/api

tentar obter a lista de géneros, usando mais especificamente o método track.getInfo[98] . No caso de insucesso, existe um conjunto de erros registados por esse mesmo método que podem ocorrer, como por exemplo o número de pedidos ser excessivo ou o serviço estar temporariamente indisponível e assim será produzida uma mensagem igual à anterior para o microsserviço Manager, mas com o resultado de acordo com o erro gerado. No caso de o procedimento ser bem-sucedido, o fluxo da figura segue pelo ramo de baixo onde enviará a lista de géneros obtidos. Ambos os tipos de resposta carregam sempre o identificador do vídeo porque é através deste que é possível identificar o respetivo documento na base de dados para ser atualizado com as informações do género.

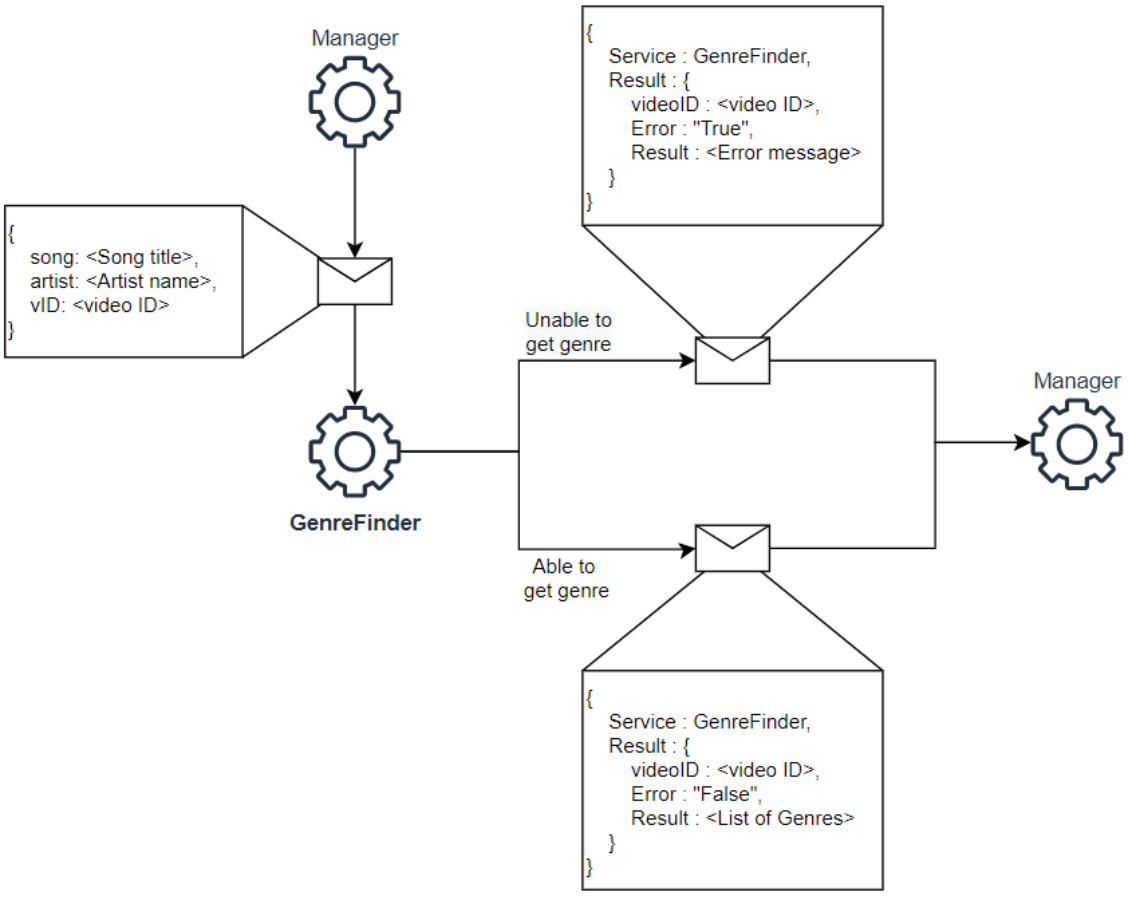


Figura 50 - Lógica do microsserviço GenreFinder

[98] https://www.last.fm/api/show/track.getInfo

5.6. LyricsExtractor

O objetivo deste microsserviço é obter a lírica de uma música para se conseguir adquirir também a classificação emocional desta componente, resultando num sistema bi-modal assim como estudado em alguns trabalhos de investigação da área. Este microsserviço foi desenvolvido utilizando a linguagem de programação Python e está também ligado ao intermediador de mensagens utilizando a biblioteca Pika. Consome a fila de mensagens *Lyrics* e, assim como o microsserviço *GenreFinder*, também aqui as mensagens contêm três parâmetros: o identificador do vídeo, o título da música e o nome do artista caso tenham sido possíveis de obter.

A primeira validação realizada é avaliar se a mensagem consumida contém os campos título da música e nome do artista porque não é possível obter a lírica de uma música sem os mesmos. Na ausência destes, é produzida uma mensagem para a fila *Management* seguindo a estrutura habitual já descrita anteriormente, com a informação de que a música não foi encontrada e o identificador do vídeo, como mostra a Figura 51. Se os meta-dados do áudio forem recebidos, será então realizada a tentativa de obter a sua lírica utilizando a API da Genius, como foi descrito na subsecção 2.9, onde foi criada uma conta na sua plataforma para ter acesso a uma chave e utilizar a mesma através da biblioteca para Python *lyricsgenius*[99] e tentar obter a lírica dos áudios com os meta-dados que foram adquiridos. Se a base de dados da Genius não contiver a música que se procura, quer seja porque realmente não existe ou porque o nome da música ou do artista eram incorretos, será produzida uma mensagem exatamente igual à anterior a informar que não foi possível encontrar a lírica para este áudio e o identificador do vídeo para identificar o seu registo na base de dados e para inserir este texto de error no campo da lírica. Caso seja possível de encontra a lírica, a mesma é registada num ficheiro de texto, onde o nome do ficheiro é o título da música, que é armazenado numa diretoria partilhada pelos microsserviços e é produzida uma mensagem a incluir o nome do ficheiro armazenado que contém a lírica para depois serem extraídas

[99] https://lyricsgenius.readthedocs.io/en/master/

caraterísticas sobre este ficheiro, a lírica obtida para inserir no respetivo registo da base de dados e o identificador do vídeo para identificar esse mesmo registo.

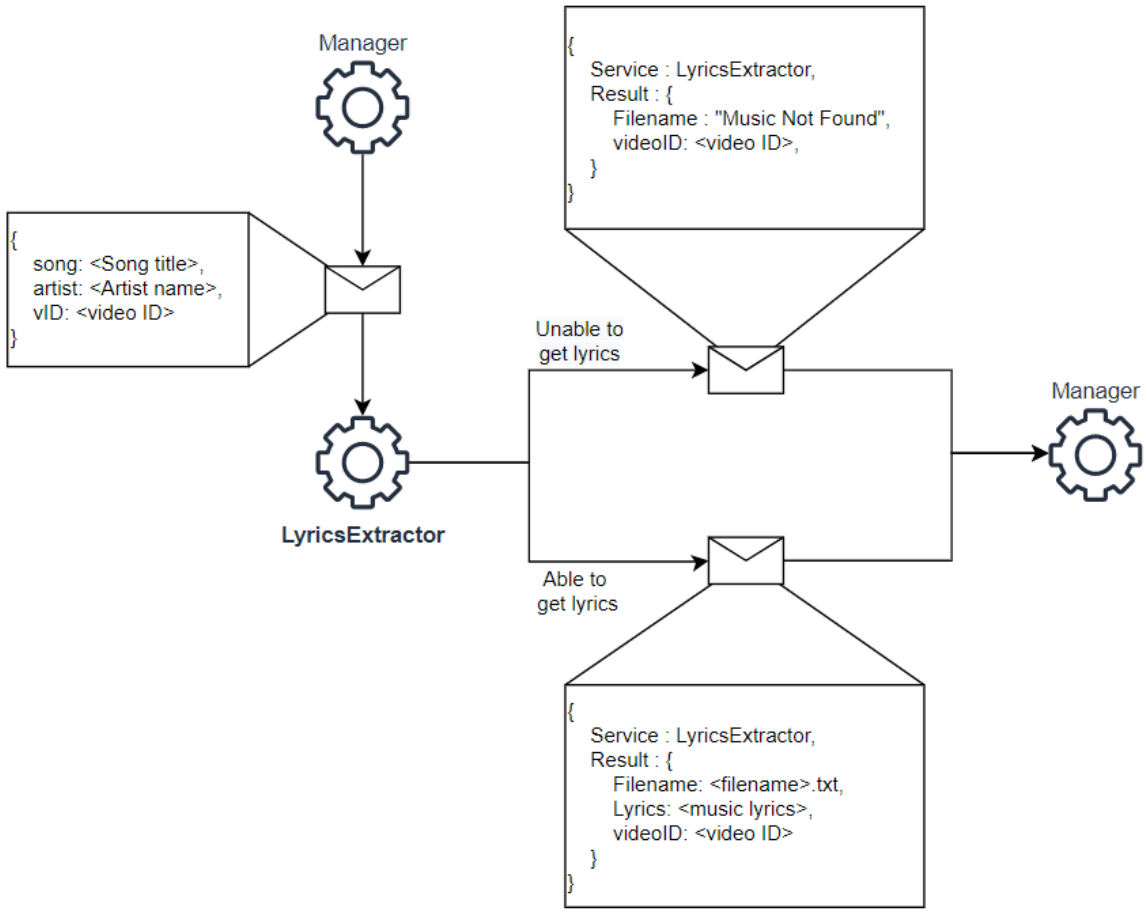


Figura 51 - Lógica do microsserviço LyricsExtractor

5.7. LyricsFeaturesExtractor

O microsserviço LyricsFeaturesExtractor tem por objetivo extrair caraterísticas da lírica e foi desenvolvido em Python. Está ligado ao intermediador de mensagens através da biblioteca Pika, consome a fila de mensagens *lyricsFeatures* e produz para a fila *Management*. As mensagens por este consumidas contêm o identificador do vídeo e o nome do ficheiro de texto a processar assim como ilustrado na Figura 52.

Como descrito na secção 2.6.2, a utilização da lírica neste sistema está num estágio inicial e, portanto, foi apenas analisada uma *framework* para extração de caraterísticas da lírica. A

framework em questão pode ser utilizada através de uma interface gráfica ou através de um executável em linha de comandos. A opção escolhida foi utilizar o executável e, sendo o microsserviço desenvolvido em Python, foi utilizado o módulo *subprocess*[100] para criar processos e correr a instrução do executável como se fosse em linha de comandos. O executável está a extrair todas as caraterísticas possíveis pela *framework*, são exatamente 198 caraterísticas que se baseiam na semântica, estrutura e no conteúdo e estas são extraídas para um ficheiro Excel com formato csv (*Comma-separated values*). De seguida é necessário ler o ficheiro Excel para avaliar se existem valores nulos e substitui-los por zero porque SVM não lida muito bem com a falta de dados e pode impactar negativamente o modelo (Stewart et al., 2018) e assim optou-se por substituir pelo valor zero. Por fim é criado um objeto com todas as características extraídas porque se pretende guardar as caraterísticas na base de dados NoSQL nesse formato e juntamente com o identificador do vídeo constitui a mensagem produzida para a lista *Management*, como se ilustra de seguida.

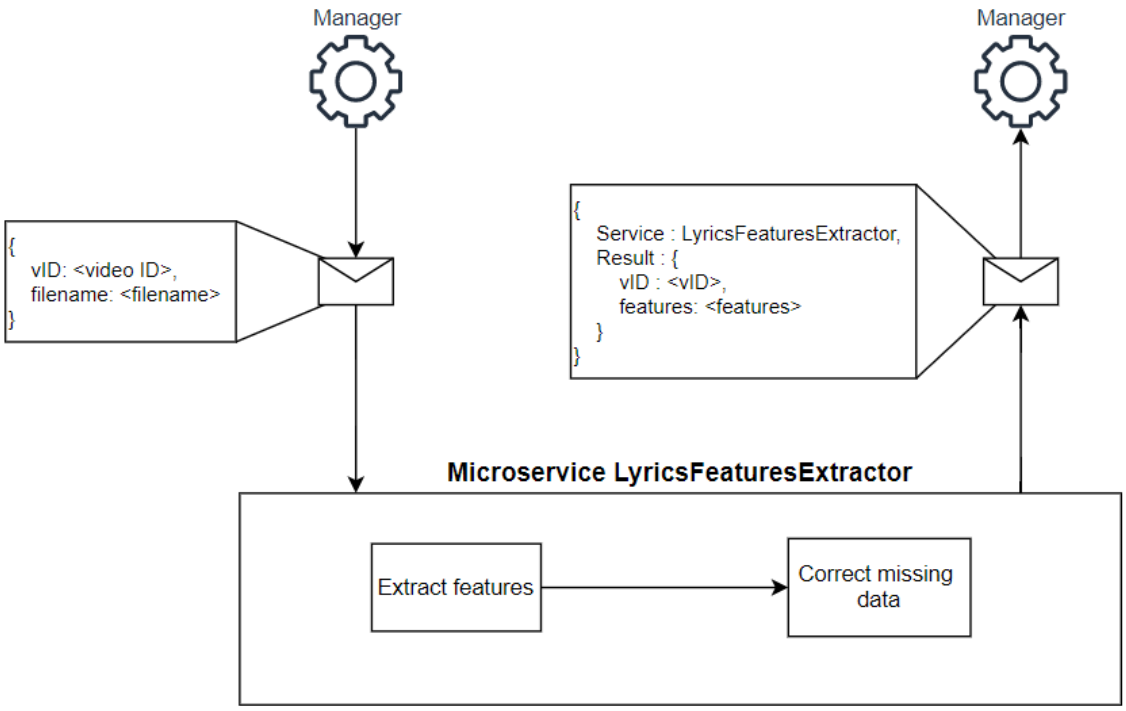


Figura 52 - Lógica do microsserviço LyricsFeaturesExtractor

[100] https://docs.python.org/3/library/subprocess.html

5.8. SourceSeparation

Este microsserviço tem como finalidade separar fontes do áudio (elementos musicais). Foi desenvolvido tendo em conta algumas experiências que se vêm a realizar na área de MER porque existe o conceito de que por vezes é possível distinguir se uma música é 'calma' ou 'triste' através de determinadas caraterísticas aplicadas à componente vocal, mas esta propriedade perde-se quando as componentes são misturaras (R. Panda et al., 2020b), portanto pretende-se que este microsserviço separe as fontes dos áudios para se poder ter classificadores associados a cada componente.

O microsserviço foi escrito em Python e mais uma vez está também ligado ao intermediador de mensagens utilizando a biblioteca Pika, consumindo a fila de mensagens *Separate*. As mensagens por este consumidas contêm apenas o identificador do vídeo, porque como foi explicado no microsserviço VidExtractor, os áudios são sempre descarregados para a mesma diretoria, que é partilhada entre microsserviços e, portanto, este microsserviço precisa apenas de saber qual o nome do ficheiro sobre o qual vai aplicar a separação de elementos porque a diretoria já é por ele conhecida.

Como é mostrado na Figura 53, o passo inicial é obter o ficheiro de áudio da diretoria partilhada que tem o nome igual ao identificador do vídeo enviado na mensagem. De seguida, para realizar a separação de fontes está a ser utilizada a biblioteca Spleeter[101] com um modelo por eles pré-treinado que exerce a separação em duas fontes: a voz e o acompanhamento. Seria também possível separar em quatro ou cinco fontes, mas estando esta componente ainda num estágio inicial, só fazia sentido separar em duas fontes de momento. O método de separação necessita de dois parâmetros, o áudio de entrada e a diretoria final, sendo que o próprio já cria uma pasta com os áudios separados lá dentro, independentemente do número de fontes a separar, ficando o nome da pasta igual ao nome do ficheiro de áudio que é fornecido na entrada. Em suma, assim como é mostrado na Figura 53 pela ligação (2), após a separação é criada uma pasta na diretoria partilhada com o nome

[101] https://github.com/deezer/spleeter

do <videoID> que contém os ficheiros *accompaniment.wav* e *vocals.wav* para acompanhamento e voz, respetivamente. Depois, o ficheiro de áudio utilizado como entrada, que corresponde ao áudio original, é movido para dentro da pasta agora criada na diretoria partilhada e renomeado para *original.wav*. Assim, após realizado todo o procedimento deste microsserviço, todos os ficheiros relativos a um vídeo ficam na mesma diretoria nomeada pelo identificador do vídeo, como mostra a ligação (3) da Figura 53. Por fim é produzida uma mensagem para a fila de mensagem *Management* com o identificador do vídeo para que o próximo microsserviço conheça qual a pasta onde estão os ficheiros do vídeo.

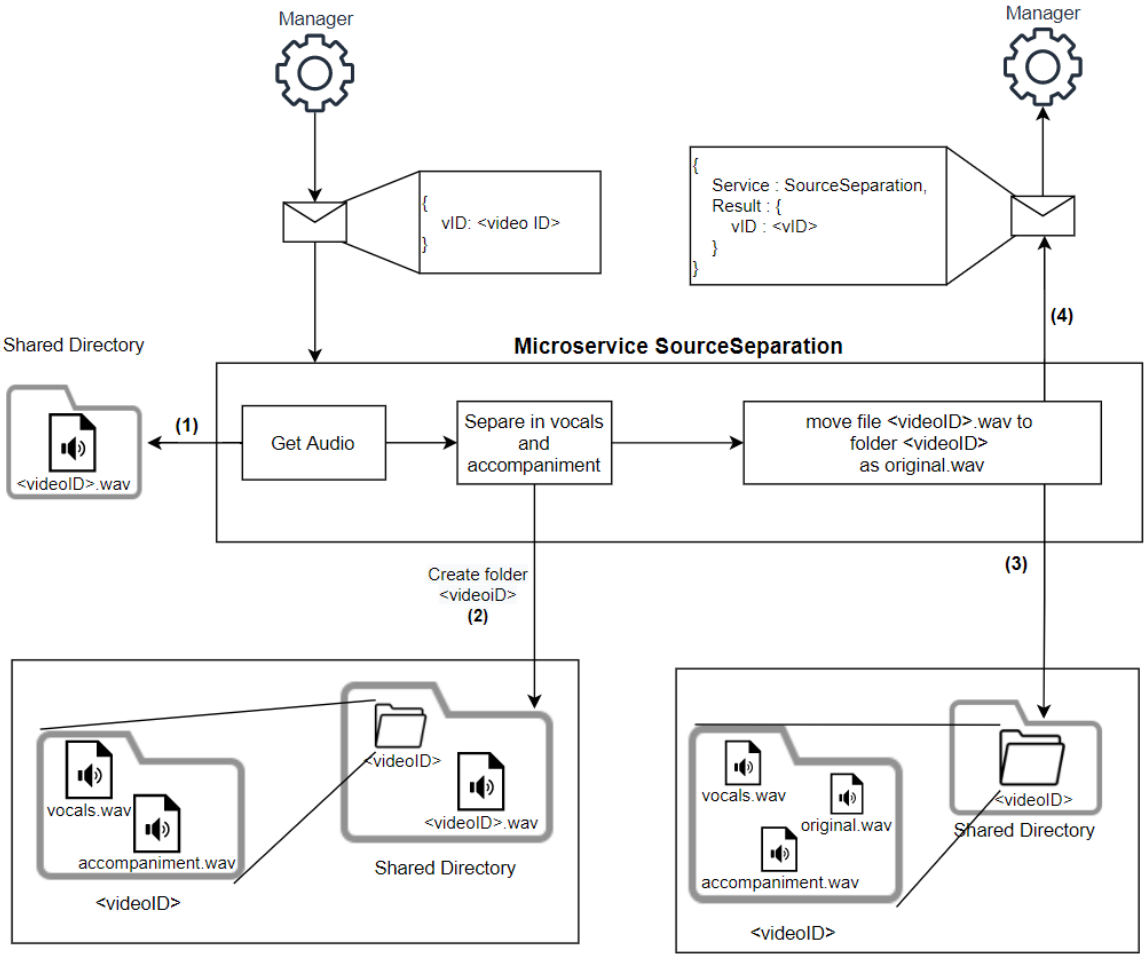


Figura 53 - Lógica do microsserviço SourceSeparation

5.9. Segmentation

O microsserviço Segmentation tem por intuito segmentar ficheiros áudio e alterar duas propriedades dos áudios, que são a taxa de amostragem e o número de canais em uso. Foi desenvolvido em Node.js, liga-se ao intermediador de mensagens através da biblioteca amqplib e consome a fila de mensagens *Segmentation* de onde recebe mensagens com o identificador de um vídeo como conteúdo.

Este microsserviço, no estado atual do sistema, deve ser executado após o microsserviço SourceSeparation e irá segmentar os ficheiros que foram originados por esse, e é por isso que necessita de saber qual o identificador do vídeo, para ir à diretoria partilhada segmentar os ficheiros *accompaniment.wav*, *original.wav* e *vocals.wav* que estão dentro da pasta com aquele identificador. Os ficheiros resultantes da segmentação têm a duração fixa de 30 segundos, para seguir a literatura, exceto quando está a gerar os últimos ficheiros em que a duração é igual à duração que resta até ao final da música. Os excertos são registados com uma sobreposição (*overlapping*) de 15 segundos e a Figura 54 serve de exemplo ao fazer a segmentação de um ficheiro de 50 segundos com o nome *original*, onde são gerados quatro novos ficheiros, em que os dois primeiros têm duração de 30 segundos, mas o *original_3.wav* já só tem duração de 20 segundos e o *original_4.wav* tem duração de 5 segundos porque é a duração que resta até ao final para segmentar.

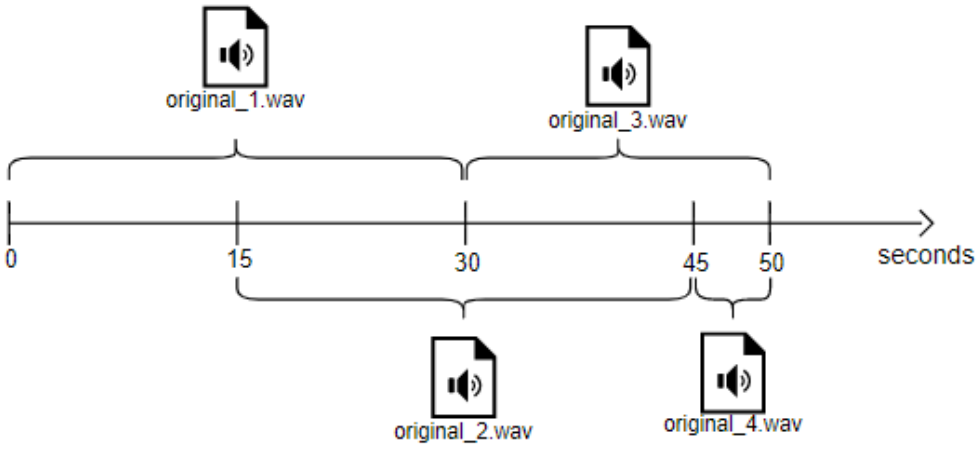


Figura 54 - Exemplo de segmentação para áudio de 50 segundos

Para fazer a segmentação dos áudios está a ser utilizado mais uma vez a biblioteca fluent-ffmpeg, que para além de segmentar os ficheiros permite também manipular as propriedades do áudio e por isso antes de guardar os novos ficheiros, estes são definidos para utilizar apenas um canal (*mono channel*) em vez dos dois canais de quando é descarregado (*stereo channel*) e a taxa de amostragem é reduzida para 22 500 amostras. Os ficheiros são guardados com estas propriedades para "reduzir a carga intensiva dos algoritmos de extração computacionais" (R. E. S. Panda, 2019). Os ficheiros a segmentar são o *accompaniment.wav*, *original.wav* e *vocals.wav*, originados no microsserviço SourceSeparation, e os novos ficheiros segmentados vão ser guardados na mesma diretoria, com os seus nomes a seguirem a nomenclatura do ficheiro completo seguido de _ (*underscore*) e um contador que vai sendo incrementado à medida que os ficheiros vão sendo gerados, como foi no exemplo da Figura 54, onde o ficheiro se chamava original. No fim de gerados e guardados todos os ficheiros, será produzida uma mensagem para a fila *Management*, como mostra a Figura 55, com o identificador do vídeo e o número de ficheiros originados, que será depois utilizado quando for necessário classificar os mesmos e garantir que são todos classificados.

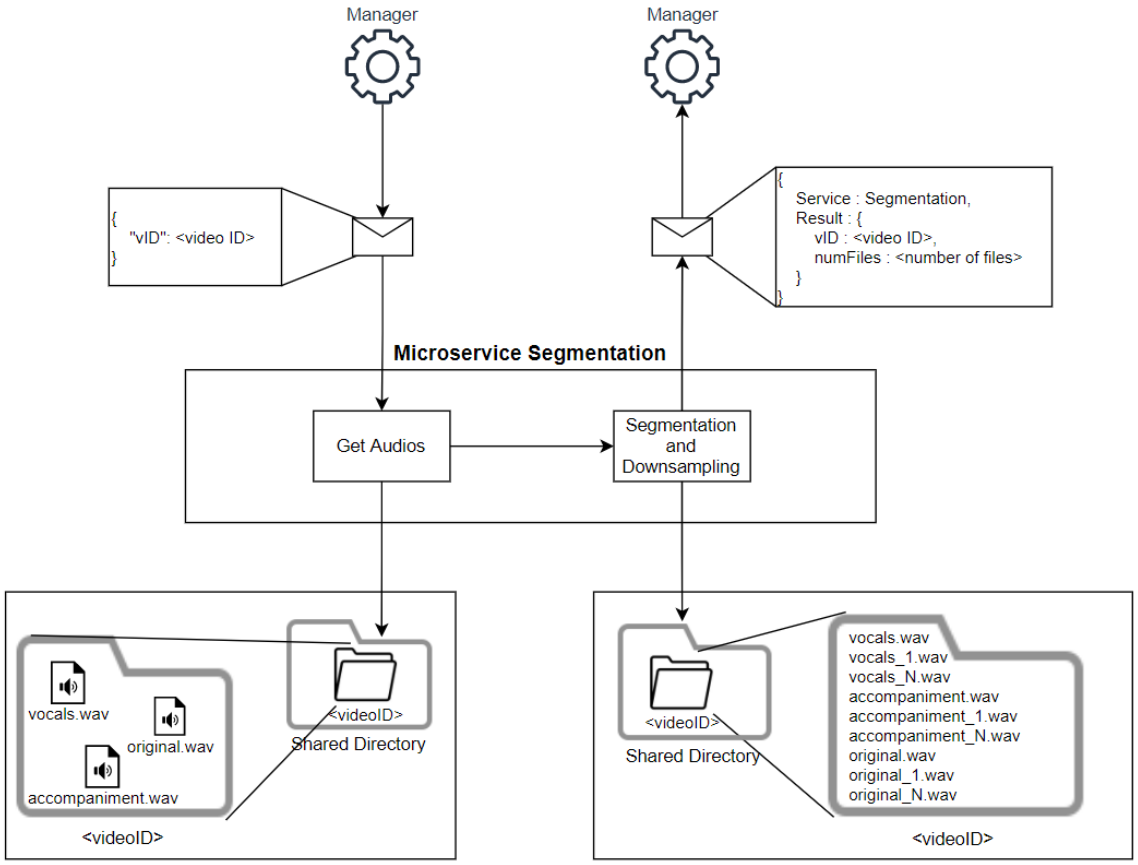


Figura 55 - Lógica do microsserviço Segmentation

5.10. AudioFeaturesExtractor

Este microsserviço tem por objetivo extrair caraterísticas dos áudios. Foi escrito em Python, liga-se ao intermediador de mensagens através da biblioteca Pika, de onde consome a fila *musicFeatures* e produz para a fila *Management*. A mensagem por este consumida contem o identificador do vídeo, a localização do ficheiro para efetuar a extração de caraterísticas, a fonte musical a que pertence o excerto e uma variável (*flag*) para indicar se o ficheiro a extrair caraterísticas é o último referente ao vídeo.

A extração de caraterísticas está a ser feita utilizando a biblioteca Essentia[102] com o método MusicExtractor[103] que recebe como entrada o ficheiro de áudio e como parâmetro as caraterísticas que se pretende extrair. As caraterísticas extraídas são as mesmas já definidas na secção 3.2.

Existem casos em que não é possível extrair caraterísticas dos áudios porque os mesmos podem ser silenciosos. Por exemplo um vídeo que só tenha componente instrumental ficará silencioso ao fazer a separação de fontes para a componente de voz e assim não é possível extrair caraterísticas do mesmo. Quando isto acontece é produzida uma mensagem com o identificador do vídeo, a fonte musical, a variável a indicar que se trata ou não do último ficheiro de um vídeo a extrair caraterísticas e a mensagem de que ocorreu um erro ou ficheiro pode ser silencioso, assim como mostra a Figura 56.

Se for possível extrair as caraterísticas, é necessária atenção às caraterísticas de escala e chave da peça musical, onde está a ser associado o valor 0 ao acorde menor (*minor*) e 1 ao acorde maior (*major*) no caso da escala, e está a ser utilizada a notação para valores inteiros na chave da peça como definido em *Pitch Class*[104]. Depois é ainda necessário avaliar se existem valores nulos ou infinitos entre os valores extraídos e devem ser substituídos por zero. Por fim é então produzida uma mensagem com as caraterísticas extraídas, a fonte musical, o identificador do vídeo e a variável se é ou não o último ficheiro a extrair caraterísticas daquele vídeo.

[102] https://essentia.upf.edu/index.html
[103] https://essentia.upf.edu/reference/std_MusicExtractor.html
[104] https://en.wikipedia.org/wiki/Pitch_class

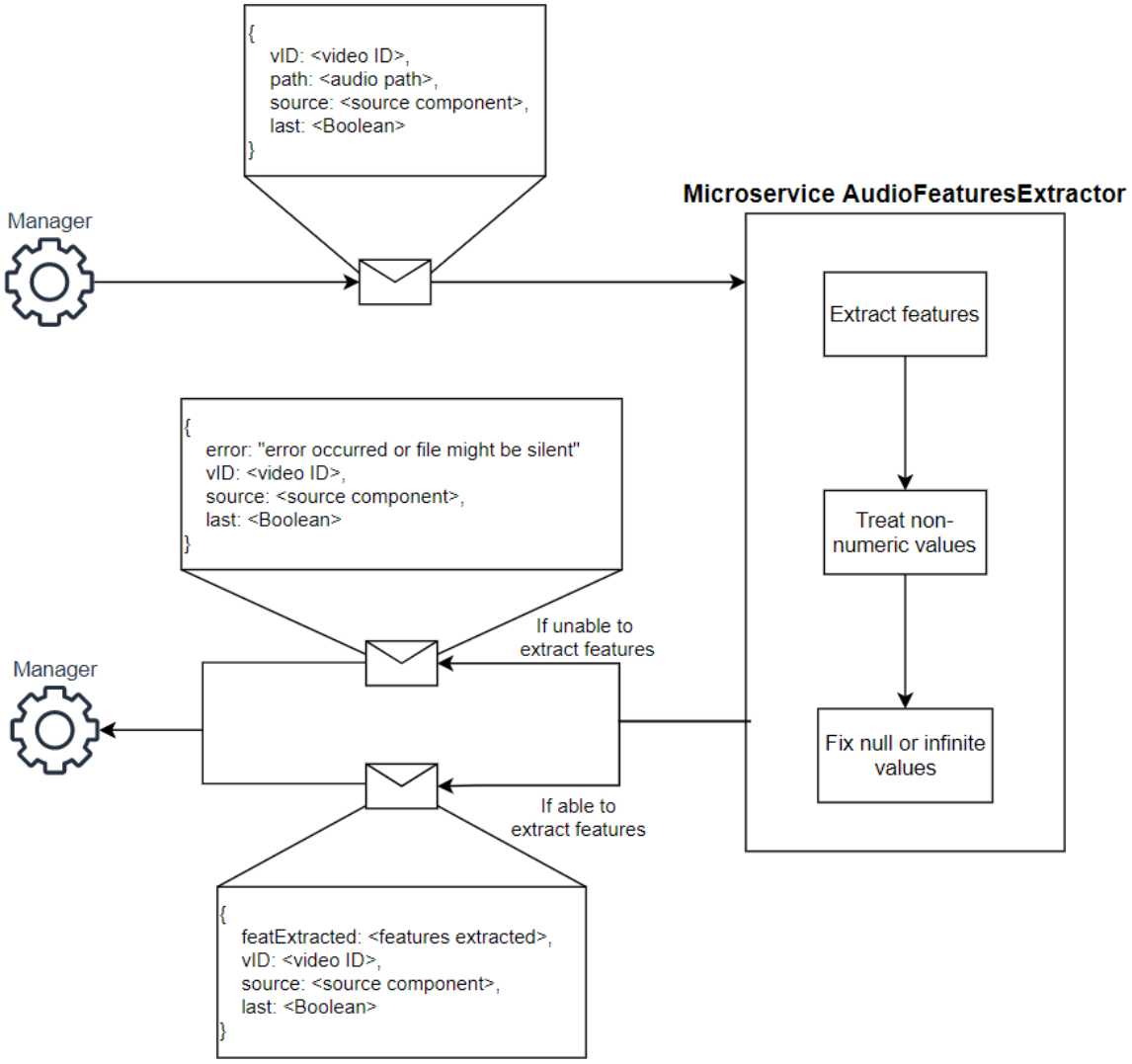


Figura 56 - Lógica do microsserviço AudioFeaturesExtractor

5.11. Classifier

O microsserviço Classifier tem por objetivo obter a emoção através de caraterísticas. Foi desenvolvido em Python e está ligado ao intermediador de mensagens através da biblioteca Pika de onde consome a fila de mensagens *classifyMusic* e produz para a fila *Management*. As mensagens que este consome contêm o identificador do vídeo, as caraterísticas extraídas para obter classificação e a fonte musical a que este corresponde. Atualmente existe apenas um único microsserviço Classifier para poupar recursos e não ter cinco *containers* ligados sendo uma tarefa simples, mas podem ser divididos num trabalho futuro.

No caso de no campo das características enviadas se encontrar a palavra *error*, significa que não tinha sido possível extrair características e também não será possível obter emoção, pelo que será produzia uma mensagem com o identificador do vídeo, a fonte musical e o campo da emoção tem o valor *Cannot get emotion*, como ilustrado na Figura 57. Em caso contrário significa que foram recebidas caraterísticas corretas e dependendo do valor enviado no campo fonte musical, será carregado o respetivo classificador, *scaler* e nome de caraterísticas a utilizar. Entre todas as características recebidas, vão ser utilizadas apenas aquelas cujo nome se encontra entre a lista de características que foi carregado, ou seja, que tinham sido utilizadas para treinar aquele modelo. De seguida, os valores das caraterísticas são transformados utilizando o *scaler* e finalmente é obtida a classificação que varia entre os valores de 1 a 4, que foram as etiquetas utilizadas no processo de treino, correspondo aos quadrantes em que 1- Alegre, 2 - Tensa, 3 – Triste e 4 – Calma. Para terminar, será então produzida uma mensagem que contém o identificador do vídeo, a fonte musical e a emoção obtida.

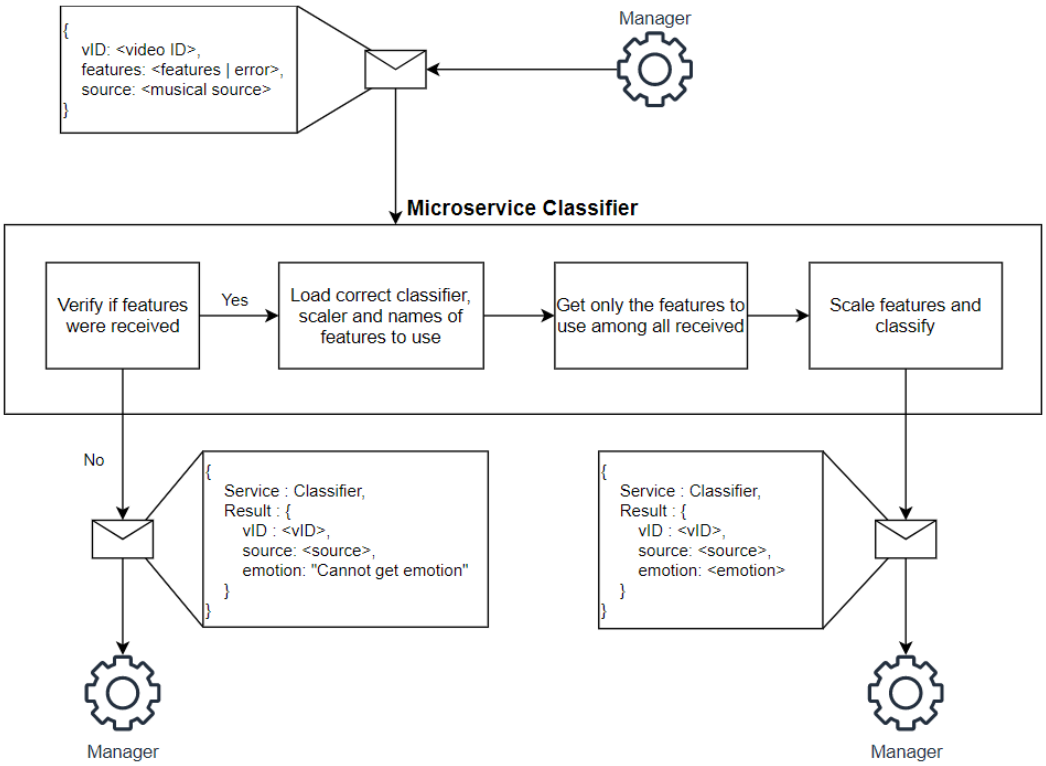


Figura 57 - Lógica do microsserviço Classifier

Capítulo 6
Conclusão

A área de reconhecimento de emoções em música tem recebido uma atenção crescente ao longo dos anos e é expectável que assim continue tendo em conta a importância que a música tem para o ser humano. O facto de que todos os dias é produzida e disponibilizada uma imensa quantidade de músicas, mais recentemente em plataformas de *streaming* de áudio, torna cada vez mais a catalogação de música um processo complexo, assim como a sua filtragem nas pesquisas devido aos típicos mecanismos de pesquisa baseados em meta-dados definidos manualmente. É também facto de que as músicas transmitem emoções que se refletem a nível físico e psicológico no ouvinte e por isso tem também sido bastante explorada e utilizada para fins medicinais e é algo que com certeza motiva os investigadores na área a desenvolver sistemas robustos que possam ajudar nesta catalogação.

O objetivo deste trabalho consistiu em reorganizar uma prova de conceito e integrar novos serviços para obter informações emocionais em música, para que pudesse assim ser utilizada uma caraterística da própria música, a sua emoção, no processo de catalogação, surgindo assim uma nova possibilidade de pesquisa. A solução desenvolvida tornou-se num sistema bi-modal, baseado nas componentes de áudio e lírica, sendo que a lírica por estar num estágio inicial não foi tão focada a nível de extração de caraterísticas, já a componente de áudio sofreu várias novas integrações que são consideradas vantajosas pela literatura. Tornou-se assim um sistema bastante mais robusto e escalável, fazendo uso de tecnologias bastante atuais para as novas integrações, tentando ao máximo focar-se na redução do tempo de espera por parte do utilizador, tendo em conta que o processo de classificação é bastante pesado computacionalmente. Este sistema será implementado sobre uma solução de *containers* num trabalho paralelo e, portanto, torna-se bastante resiliente e ainda mais tolerante a falhas por exemplo, para além da já utilização de filas de mensagens que permite que não exista perdas de mensagens.

Os resultados obtidos pelos classificadores foram bastante satisfatórios, variando as suas taxas de acertos entre os 62% e 70%, onde o classificador com melhor resultado foi aquele que utiliza todas as fontes musicais. Foi também curioso perceber que músicas que

transmitem emoções alegres são mais facilmente classificadas do que as músicas que transmitem emoções calmas e ainda que as características de baixo-nível estão em número muito maior após a eliminação de caraterísticas no processamento dos dados de entrada.

Existem vários pontos deste sistema que ainda podem sofrer melhorias e novas funcionalidades num trabalho futuro, como por exemplo na extração de caraterísticas da lírica, mas espera-se que já tenha umas boas bases para que possa ser útil na área de reconhecimento de emoções em música, até pela pequena quantidade de plataformas atualmente a demonstrar este conceito.

6.1. Trabalho futuro

Este sistema consiste numa restruturação do sistema desenvolvido na prova de conceito (António, 2019) e inclui uma série de caraterísticas que já tinham sido definidas no mesmo. Mas existem sempre funcionalidades interessantes que se podem adicionar para tornar o sistema mais robusto, entre elas seria interessante implementar no futuro:

- Averiguar a utilização de *frameworks* que façam a extração de caraterísticas de mais alto nível como as identificadas em (R. Panda et al., 2020a);
- Averiguar se é vantajoso a utilização de *Deep Learning* (Aprendizagem Profunda) quando existirem conjuntos de dados que suportem esta possibilidade;
- Procurar outras *frameworks* para extração de caraterísticas da lírica ou analisar a possibilidade de extrair uma maior quantidade de caraterísticas com a *framework* em atual utilização;
- Separar o áudio original em mais fontes musicais, como por exemplo: guitarra, bateria, voz e restante instrumental;
- Permitir a classificação de informação que existe nos próprios vídeos com base em visão computacional;
- Utilização de outras plataformas como origem de dados, sem ser apenas o YouTube;
- A utilização do género como característica para classificação;
- A possibilidade de gerar listas de reprodução com base nos resultados, exportando-as para listas do Spotify.

Referências

António, R. M. (2019). *Microsserviços para Reconhecimento de Emoção em Música* [Instituto Politécnico de Tomar]. http://hdl.handle.net/10400.26/31444

de Witte, M., Pinho, A. da S., 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*. https://doi.org/10.1080/17437199.2020.1846580

Défossez, A., Usunier, N., Bottou, L., & Bach, F. (2019). *Music Source Separation in the Waveform Domain*. https://hal.archives-ouvertes.fr/hal-02379796

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media. https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

Ghofrani, J., & Lübke, D. (2018). Challenges of microservices architecture: A survey on the state of the practice. *CEUR Workshop Proceedings*, *2072*, 1–8. http://ceur-ws.org/Vol-2072

Hennequin, R., Khlif, A., Voituret, F., & Moussallam, M. (2020). Spleeter: a fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software*, *5*(50), 2154. https://doi.org/10.21105/joss.02154

Hira, Z. M., & Gillies, D. F. (2015). A Review of Feature Selection and Feature Extraction Methods Applied on Microarray Data. *Advances in Bioinformatics*, *2015*, 1–13. https://doi.org/10.1155/2015/198363

Juslin, P. N. (2013). What does music express? Basic emotions and beyond. *Frontiers in Psychology*, *4*(SEP). https://doi.org/10.3389/fpsyg.2013.00596

Kim, Y. E., Schmidt, E. M., Migneco, R., Morton, B. G., Richardson, P., Scott, J., Speck, J. A., & Turnbull, D. (2010). *Music Emotion Recognition: A State of the Art Review*. https://www.semanticscholar.org/paper/State-of-the-Art-Report%3A-Music-Emotion-Recognition%3A-Kim-Schmidt/a977808a4a56d6adf0cd0bb011638ed8d2d97b99

Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques. *Informatica (Ljubljana)*, *31*(3), 249–268. https://doi.org/10.31449/inf.v31i3.148

Laurier, C., Grivolla, J., & Herrera, P. (2008). Multimodal music mood classification using audio and lyrics. *Proceedings - 7th International Conference on Machine Learning and Applications, ICMLA 2008*, 688–693. https://doi.org/10.1109/ICMLA.2008.96

Malheiro, R., Panda, R., Gomes, P., & Paiva, R. P. (2016). *Emotionally-Relevant Features for Classification and Regression of Music Lyrics*. https://doi.org/10.1109/TAFFC.2016.2598569

Manilow, E., Seetharaman, P., & Pardo, B. A. (2018). *The northwestern university source separation library* (Emilia Gomez, Xiao Hu, Eric Humphrey, & Emmanouil Benetos (eds.); pp. 297–305). International Society for Music Information Retrieval. http://ismir2018.ircam.fr/doc/pdfs/37_Paper.pdf

Mathieu, B., Essid, S., Fillon, T., Prado, J., & Richard, G. (2010). *YAAFE, AN EASY TO USE AND EFFICIENT AUDIO FEATURE EXTRACTION SOFTWARE*. https://archives.ismir.net/ismir2010/paper/000075.pdf

Mcfee, B., Raffel, C., Liang, D., Ellis, D. P. W., Mcvicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and Music Signal Analysis in Python. In *PROC. OF THE 14th PYTHON IN SCIENCE CONF*. http://conference.scipy.org/proceedings/scipy2015/pdfs/brian_mcfee.pdf

Moffat, D., Ronan, D., & Reiss, J. D. (2015). *AN EVALUATION OF AUDIO FEATURE EXTRACTION TOOLBOXES*. http://qmro.qmul.ac.uk/xmlui/handle/123456789/13075

Panda, R. E. S. (2019). *Emotion-based Analysis and Classification of Audio Music* [Faculdade de Ciências e Tecnologia da Universidade de Coimbra]. http://hdl.handle.net/10316/87618

Panda, R., Malheiro, R. M., & Paiva, R. P. (2020a). Audio Features for Music Emotion Recognition: a Survey. *IEEE Transactions on Affective Computing*, 1–1. https://doi.org/10.1109/TAFFC.2020.3032373

Panda, R., Malheiro, R., & Paiva, R. P. (2020b). Novel Audio Features for Music Emotion Recognition. *IEEE Transactions on Affective Computing*, *11*(4), 614–626. https://doi.org/10.1109/TAFFC.2018.2820691

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot. In *Journal of Machine Learning Research* (Vol. 12). http://scikit-learn.sourceforge.net.

Stewart, T. G., Zeng, D., & Wu, M. C. (2018). Constructing support vector machines with missing data. *Wiley Interdisciplinary Reviews: Computational Statistics*, *10*(4). https://doi.org/10.1002/wics.1430

Vural, H., Koyuncu, M., & Guney, S. (2017). A Systematic Literature Review on Microservices. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 10409 LNCS* (pp. 203–217). Springer Verlag. https://doi.org/10.1007/978-3-319-62407-5_14

World Health Organization. (2020). *Mental health and psychosocial considerations during the COVID-19 outbreak*. https://www.who.int/docs/default-source/coronaviruse/mental-health-considerations.pdf

Xinyu Yang, Yizhuo Dong, & Juan Li. (2018). Review of data features-based music emotion recognition methods. *Multimedia Systems*, *24*(4), 365–389. https://doi.org/10.1007/S00530-017-0559-4

Xu, J., Li, X., Hao, Y., & Yang, G. (2014). Source Separation Improves Music Emotion Recognition. *Proceedings of International Conference on Multimedia Retrieval*, 423–426. https://doi.org/10.1145/2578726.2578784

Yang, Y.-H., & Chen, H. H. (2012). *Machine Recognition of Music Emotion: A Review*. *3*. https://doi.org/10.1145/2168752.2168754