Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Ricardo Marques António

Microsserviços para Reconhecimento de Emoção em Música

Relatório de Projeto de Mestrado

Orientado por:

Professor Renato Panda, Instituto Politécnico de Tomar
Professor Luís Oliveira, 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

“Aos que sempre me apoiaram e acreditaram em mim, um muito obrigado.”

Ricardo António

RESUMO

O reconhecimento de emoção em música (MER) é uma área de investigação que tem ganho notoriedade nas últimas décadas. Sabemos hoje que a música transmite emoções aos ouvintes, sendo este facto explorado para os mais diversos fins, como o entretenimento, cinema, desporto ou terapia. Por esta razão, investigadores na área de MER têm estudado a possibilidade de criar sistemas capazes de reconhecer de forma automática a emoção presente num sinal musical áudio. Este processo é extremamente complexo, tocando diversas áreas como a psicologia, com as taxonomias de emoção, mas também de ciência da computação, com processamento de sinal e aprendizagem automática, assim como a área da música. Desta forma, grande parte dos avanços permanece ainda fechada no meio académico, existindo poucos ou nenhuns sistemas deste tipo disponíveis.

Este projeto visou criar um protótipo sistema robusto e escalável que sirva para demonstrar o conceito de um sistema de MER. Para isso junta os vários conceitos do estado da arte na área com metodologias modernas de engenharia de *software*, originando um sistema que obtém o áudio de vídeos do YouTube, sumariza o mesmo em características e classifica-o em quatro emoções distintas: alegria, agressividade, tristeza e tranquilidade. Foram desenvolvidos microsserviços distintos que implementam estas funcionalidades, através de uma arquitetura que os torna independentes, escaláveis, robustos e de alto desempenho. Este sistema vem contribuir também para a área de MER, pois facilita a divulgação da mesma fora do meio académico.

ABSTRACT

Music Emotion Recognition (MER) is a research area that has gained notoriety in recent decades. We know today that music transmits emotions to listeners, and this fact is exploited for various purposes, such as entertainment, cinema, sports or therapy.

For this reason, researchers in the area of REM are studying the possibility of creating systems that can automatically recognize the emotion present in an audio musical signal. This is an extremely complex process, touching many areas such as psychology, with the taxonomies of emotion, but also computer science, with signal processing and automatic learning, as well as the area of music.

In this way most advances remain closed in the academic world, with few or no such systems available.

This project aimed to create a robust and scalable prototype system that serves to demonstrate the concept of a MER system. For this, it joins several state-of-the-art concepts in the area with modern software engineering methodologies, that origins a system that obtains audio from You Tube videos, summarizes it into characteristics and classifies it into four distinct emotions: joy, aggressiveness, sadness and tranquility. Distinct microservices have been developed that implement these functionalities through an architecture that makes them independent, scalable, robust and high performance.

This system also contributes to the MER area as it facilitates the divulgation outside the academic world.

AGRADECIMENTOS

Este trabalho não teria sido possível sem a presença de um conjunto de pessoas na minha vida, pelo que lhes dedico este trabalho como forma de gratidão por todo o apoio, suporte e amor que me deram durante as fases mais complicadas deste ano extremamente difícil. Para os meus pais Hélder e Irene, o meu irmão Tiago e a minha namorada Sílvia. O meu muito obrigado.

Ao Professor Doutor Renato Panda pelo suporte, incentivo e disponibilidade que sempre me ofereceu. Pelo tempo e apoio que me deu nas fases mais árduas do projeto e também durante este ano difícil, um especial obrigado.

Um obrigado também ao Professor Doutor Luís Oliveira e à professora Doutora Ana Cristina Lopes pelo apoio e disponibilidade durante este árduo ano.

Aos professores do Mestrado de Engenharia Informática do Instituto Politécnico de Tomar um obrigado pela partilha de conhecimento e disponibilidade demonstradas durante os dois anos de mestrado.

Finalmente, aos meus colegas David Bernardo, Diogo Mendes, Luís Nunes, Pedro Dias e Rafael Escudeiro um obrigado pelo apoio prestado e a total disponibilidade em ajudar.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

GLOSSÁRIO

Arousal	Excitação – energia e nível de estimulação
CD-ROM	*Compact Disk Read-Only Memory* - Memória só de leitura em Disco Compactado
Feature	Qualquer característica específica que permite descrever/diferenciar algo, por exemplo a cor dos olhos, batidas por minuto, duração de algo
Framework	Conjunto de bibliotecas e ferramentas
Playlist	Lista de Músicas
MER	*Music Emotion Recognition* - Reconhecimento de emoção em música
MIR	*Music Information Retrieval* - Recolha de informação em música
Streaming	Transmissão contínua de dados
Valence	Valência – estados positivos ou negativos relacionados com o sentimento

Capítulo 1

Introdução

A música tem estado presente na história da nossa espécie desde que há memória. Normalmente associada ao entretenimento, a música tem vários outros fins onde é utilizada e a prova disso é a sua utilização em acontecimentos marcantes, como cerimónias religiosas, aniversários, eventos desportivos ou até os hinos de países. Isto acontece porque a música é muito mais que um conjunto de alterações de pressão de ar, tendo a capacidade de transmitir várias emoções (Juslin, 2013), proporcionando bem-estar pessoal e social e tendo várias utilizações nas mais diversas áreas do dia-a-dia. Por outro lado, "nunca houve uma tão vasta coleção de música a ser criada e a ouvida diariamente" (Chen, 2012) e a principal razão deve-se à facilidade com que, atualmente, esta pode ser acedida através da Internet.

Ao longo do tempo, os métodos de gravação e distribuição de música têm vindo a evoluir, desde os formatos analógicos, como o vinil e cassetes, até aos formatos digitais compactos, como CD-ROM. Atualmente é comum ouvir música através dos serviços de *streaming* como o Spotify[1] ou YouTube[2]. Estes serviços disponibilizam catálogos de milhões de músicas dos mais diferentes géneros, nacionalidades e eras. Esta oferta gera um grande problema de excesso de informação, pois uma tão vasta coleção de músicas é difícil de consultar utilizando os métodos de pesquisa típicos, normalmente com base no título da música, artistas ou outros meta-dados definidos manualmente (Chen, 2012). Em alternativa a estas pesquisas, estes serviços dispõem de várias playlists, mas estas são por norma criadas por outros utilizadores ou geradas com base nos históricos de reproduções e não no conteúdo do sinal musical. No entanto, tendo a música um papel tão importante ao nível de transmissão de emoções, seria interessante explorar esta informação automaticamente, como por exemplo fazendo a catalogação automática (sem intervenção humana) das músicas através do sentimento que elas nos transmitem. Alguns destes serviços, por vezes,

[1] https://www.spotify.com/pt/
[2] https://www.youtube.com/

já suportam alguma pesquisa por "*moods*" (humores), permitindo por exemplo procurar músicas felizes. No entanto, por norma a base desta pesquisa não é feita através da análise do sinal de áudio, mas sim através de etiquetas que foram atribuídas por outros utilizadores (por exemplo no last.fm[3]).

Surge então a área de reconhecimento de emoções em música (MER), um campo de investigação ainda em aberto, ou seja, no qual ainda existem diversas questões em aberto, que tem vindo a ser abordado nas últimas décadas pela comunidade científica na área de extração/recuperação de informação musical (*Music Information Retrieval*, MIR). O processo em si é complexo pois lida com diferentes áreas como: psicologia (emoção) e ciência da computação (processamento de áudio e inteligência artificial). Alguns dos estudos científicos publicados, como o realizado por (Panda & Pedro Paiva), deram origem a protótipos e ferramentas que servem de conceitos. Este processo, por norma, segue um processo semelhante ao ilustrado na Figura 5 (disponível na página 16, onde é descrito de forma mais aprofundada), e passa pela análise e processamento de sinais de áudio de um conjunto de dados de treino devidamente etiquetados emocionalmente, a fim de extrair as suas características musicais, utilização de mecanismos de reconhecimento automático para treinar um modelo capaz de reconhecer padrões entre as características extraídas e respetivas etiquetas emocionais, e utilização do modelo gerado para obter (prever) a classificação emocional de uma nova música através das características extraídas desta nova música.

Sendo assim, o objetivo deste projeto consiste em desenvolver um protótipo web capaz de reconhecer emoções em música/vídeos do YouTube. Para tal, este deve seguir a abordagem típica usada no estado da arte de MER. Dada a complexidade computacional associada a determinadas tarefas (como o processamento áudio e classificação), o sistema deve ser escalável, robusto e bem organizado para que seja possível alterar cada um dos módulos facilmente. Como ilustrado na Figura 1, a abordagem recai na utilização de uma arquitetura de microsserviços isolados, em que cada um é independente e responsável

[3] https://www.last.fm/pt/

apenas por efetuar uma tarefa específica (ex. classificação). Ao contrário do processo tradicional, onde as tarefas são executadas de forma síncrona, numa sequência de eventos bem definida, estes serviços comunicam através de uma fila de mensagens que tem como base um mecanismo de produção/consumo. Assim, cada microsserviço aguarda uma nova tarefa específica para si na fila de mensagens (consumo), desencadeada através da criação de uma nova mensagem. Depois de realizar a tarefa, o microsserviço publica uma nova mensagem na fila e volta ao estado de espera de uma mensagem. Esta abordagem permite tornar cada uma das tarefas assíncronas e independentes, podendo até serem executadas várias réplicas de um mesmo microsserviço caso a sua tarefa seja mais demorada.

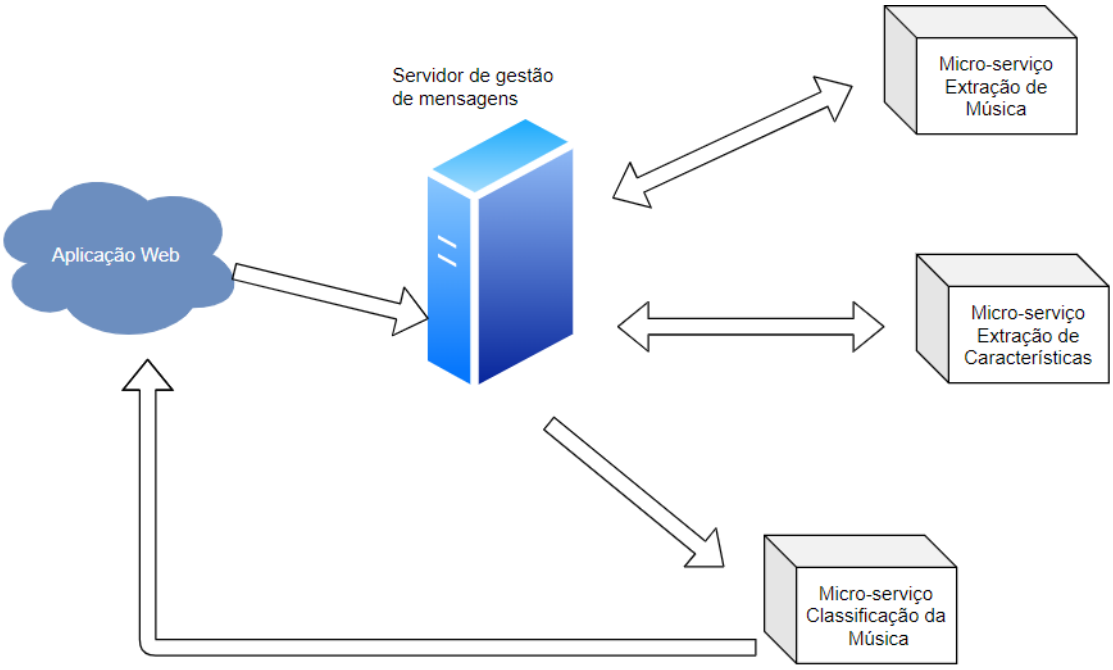

Figura 1 - Arquitetura Geral da Solução

1.1. Organização do Relatório

O presente relatório está organizado da seguinte forma: o capítulo 1 apresenta uma introdução de alto nível do trabalho realizado, no capítulo 2 é abordado o estado de arte na área e descrito o processo de reconhecimento emocional em música. De seguida, o capítulo

3 descreve a arquitetura do sistema implementado e as tecnologias utilizadas. No capítulo 4 apresentamos a informação relativa ao processo de implementação do sistema. Por fim, o capítulo 5 finaliza este trabalho, traçando a conclusão e algumas ideias para trabalho futuro.

Capítulo 2
Estado da Arte

Nesta secção é apresentada uma revisão do estado da arte na área do reconhecimento emocional em música. Nesse sentido, a secção 2.1 serve para dar um contexto do panorama atual, seguida da descrição de algumas das plataformas de *streaming* existentes no mercado de forma a perceber como é feita a procura de músicas, na secção 2.2. São também apresentados alguns protótipos desenvolvidos no âmbito de projetos de investigação que implementam funcionalidades de MER. Na secção 2.3 são abordadas as diferentes taxonomias de emoção, ou seja, os métodos propostos e aceites pela comunidade científica para identificar e representar as várias emoções. Na secção 2.4 é exposto o processo aprendizagem computacional relacionada com MER. A aprendizagem computacional é um processo estruturado que passa por três partes distintas:

- Obtenção de um conjunto de dados verdadeiros (*ground truth*), isto é, escolha de um *dataset*;
- Extração de características musicais a partir do sinal áudio, realizado através de algumas *frameworks* de tratamento do sinal de áudio;
- Reconhecimento de padrões entre as características extraídas do conjunto de dados e respetivas etiquetas, para a classificação futura de novos exemplos (Panda R. E., 2019). Este processo recorre a algoritmos de aprendizagem de máquina e bibliotecas que os aplicam;

Diversos trabalhos, como os realizados por (Panda, Malheiro, & Paiva, 2018) e (Yang, Lin, Su, & H. Chen) têm vindo a ser apresentados seguindo esta abordagem, utilizando não apenas sinais de áudio, mas também texto das líricas, como (Malheiro, Panda, Gomes, & Pedro Paiva), variando *datasets* e classificadores. A Figura 5 (localizada na página 18) demonstra todo o processo pelo qual é necessário passar para obter um resultado. Para uma visão histórica, consultar o capítulo 3.2 de (Panda R. E., 2019).

2.1. Contexto do Panorama Atual

A área de investigação que aborda o reconhecimento de emoção em música tem vindo a crescer ao longo das últimas décadas, servindo-se do conhecimento produzido em várias áreas científicas, especialmente aquelas relacionadas com a extração e recolha de informação em sinais de áudio e a aprendizagem computacional (Panda, Malheiro, & Paiva, 2018). Os avanços neste tópico de investigação, juntamente com o rápido crescimento das plataformas de *streaming* de música, vão levando a que haja cada vez mais interesse por parte da indústria em desenvolver ferramentas que permitem a extração e recolha de informação de um sinal de áudio para auxiliar na análise e catalogação das bases de dados musicais. Estas aplicações atuais que têm como base o *streaming* de música oferecem, na sua maioria, uma catalogação baseada em meta-dados definidos manualmente, como os artistas, álbuns, géneros musicais, entre outros (Chen, 2012), excluindo aqueles originados pelo tratamento do sinal áudio em relação ao sentimento transmitido.

2.2. Plataformas de *Streaming*

Existem várias plataformas que fornecem serviços automatizados de busca e organização de músicas. Porém, como referido anteriormente, esta catalogação é normalmente feita através de etiquetas definidas manualmente por humanos, tais como o título da música, nome do álbum ou autor, sendo que mesmo nestes casos é ainda incomum a catalogação da música pela emoção transmitida. Alguns exemplos destes são listados de seguida.

2.2.1. Spotify

O Spotify é uma plataforma de *streaming* de música, com uma versão gratuita e outra paga. Está disponível para telefone, computador, tablet, entre outros, e fornece milhões de

faixas e álbuns. Permite pesquisar músicas através de metadados como o nome da música ou álbum e organizar e filtrar através ordem alfabética, do título da música, do artista ou do álbum, entre outros campos. A aplicação permite navegar pelas listas de música dos amigos, de artistas ou celebridades, e permite ainda criar uma estação de rádio ou ouvir a lista de música sugerida diariamente[4]. Esta playlist diária é baseada nos diferentes géneros de música ouvidos pelo utilizador. Existe ainda uma categoria "Mood", em que é composta por diversas *playlists* geradas pelo sistema, por exemplo "Chill total", "Hits felizes", "canta no carro"[5]. No entanto, estas categorias são por norma geradas com base em padrões de leitura dos utilizadores, género musical e outras e não necessariamente uma emoção em específico.

2.2.2. Apple Music

A Apple Music é uma plataforma de *streaming* fornecida pela Apple, disponível no iTunes e também disponível para dispositivos iOS e Android. Disponibiliza atualmente 50 milhões de músicas e a sua audição pode ser feita com o dispositivo offline através do seu download. Permite pesquisar através do nome da música ou álbum, criar listas de reprodução e ouvir as listas de músicas que são sugeridas pela Apple Music, baseadas no que o utilizador tem ouvido. Também é possível criar um perfil, seguir amigos e ouvir as *playlists* que eles partilham. A aplicação permite ainda seguir as letras da música e ver as músicas mais tocadas. Este serviço tem três planos de pagamento, sendo que cada um é gratuito durante os três primeiros meses[6].

[4] https://www.spotify.com/pt/about-us/contact/
[5] https://open.spotify.com/genre/mood-page
[6] https://www.apple.com/pt/apple-music/

2.2.3. YouTube Music

YouTube Music é uma plataforma desenvolvida pelo YouTube para desktop, iOS e Android, que fornece "álbuns oficiais, músicas, vídeos, remixes, performances ao vivo e muito mais"[7]. Esta plataforma permite a pesquisa de música através de álbuns, nomes de artista, nomes da música e até pesquisa pela letra da música ou sua descrição. Este processo de pesquisa através da letra ou descrição é realizada através de inteligência artificial[8].

2.2.4. Tidal

Tidal é uma plataforma pertencente aos artistas: Alicia Keys, Arcade Fire, Beyoncé, Calvin Harris, Claudia Leite, Clifford "T.I" Harris, Coldplay, Daft Punk, Deadmau5, Jack White, Jason Aldean, J. Cole, Kanye West, Madonna, Nicki Minaj, Rihanna, Jay Z, Damian Marley, Indochine, Lil Wayne e Usher. Esta plataforma tem como objetivo "aproximar os artistas dos seus fans através de conteúdo único e experiências exclusivas"[9]. A Tidal tem 60 milhões de músicas e acima de 250 mil vídeos de alta qualidade. Permite a pesquisa de músicas através do nome, álbum, artista ou até algumas palavras-chave presentes na música[10].

2.2.5. MOODetector

Para além dos serviços de *streaming* comerciais existentes hoje no mercado, existem também alguns protótipos de sistemas MER disponibilizados por alguns grupos de

[7] https://music.youtube.com/
[8] https://music.youtube.com/
[9] https://tidal.com/whatistidal
[10] https://tidal.com/whatistidal

investigação. Estas são por norma pequenas provas de conceito com funcionalidade limitada, que foram criadas por forma a mostrar ao público em geral as possibilidades que o campo de MER oferece. Um desses casos é o MOODetector, uma aplicação protótipo desenvolvida pela equipa de MIR do Centro de Investigação em Informática e Sistemas da Universidade de Coimbra, reconhecendo a emoção de uma música e deteção de variações da emoção desta ao longo do tempo. O protótipo utiliza para extração de características a biblioteca Marsyas (introduzida na secção 0). "Realizado para ser utilizado como um típico reprodutor de música, permite a procura de música instantaneamente através do nome da música ou artista, organização de músicas através do nome da música, artista, valores de "*arousal*" e "*valence*" e permite ainda observar estatísticas relacionadas com as músicas já analisadas" (Cardoso, Panda, & Pedro Paiva). Além destas funcionalidades todas, a biblioteca faz detenção de humor nas músicas ou ao longo da música, estimando os valores de "*arousal*" e "*valence*". Esta ferramenta sofre, no entanto, de alguns problemas de estabilidade de desempenho, estando o mecanismo de classificação emocional ainda a ser corrigido numa nova versão de nome MOODetector Reloaded.

2.2.6. Shazam

Para além da área de MIR existem hoje diversas aplicações que fazem abordagens semelhantes com outros fins. Um dos exemplos mais conhecidos é o Shazam, uma aplicação disponível para dispositivos móveis que faz identificação automática de músicas com base em apenas alguns segundos. O método funcional do Shazam é baseado na ideia de criar uma impressão digital para cada música. O processo de criação da impressão digital é o processo no qual se obtém *hashs* de *tokens* de reprodução. Depois, a impressão digital resultante, é procurada na base de dados de músicas até serem encontradas algumas impressões parecidas. Essas impressões são avaliadas e a música resultante é que a tem mais pontos comuns com a impressão digital recolhida[11].

[11] http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

2.2.7. Considerações Finais sobre Plataformas de *Streaming*

Como verificado, existem várias plataformas de *streaming* de música e todas estas têm ajudado no rápido desenvolvimento de novos mecanismos de MIR. No entanto, é difícil encontrar alguma aplicação comercial que forneça um serviço idêntico ao que se pretende com este trabalho, mesmo os sistemas que aplicam algum reconhecimento de emoção em música são por norma conceitos académicos e pouco estáveis. O principal objetivo destas plataformas passa pela disponibilização da música ou de vídeo com boa qualidade, tentando agradar ao utilizador pela sugestão *playlists* mais específicas. Devido a isso, é normal que no futuro venham a incluir também elas mecanismos de pesquisa emocional.

Atualmente, estes serviços utilizam etiquetas definidas manualmente, como os nomes dos artistas, títulos da música, géneros musicais, álbuns. Em alternativa, oferecem listas de reprodução criadas por utilizadores ou com base no histórico de consumo dos seus utilizadores, excluindo na sua maioria a análise do sinal de áudio. Tendo isto em conta, este trabalho tem como objetivo a criação de um sistema protótipo que junte a parte do *streaming* de música e o tratamento de sinal de áudio disponibilizando informação emocional. Para tal segue a ideia inicial da plataforma MOODetector (pois esta já utiliza inteligência artificial para reconhecimento de emoção no sinal áudio), utilizando, no entanto, o áudio disponibilizado pelo serviço de *streaming*, seguindo uma abordagem de desenvolvimento mais robusta e que permita posteriormente incluir modelos emocionais mais fiáveis.

2.3. Taxonomias da Emoção

Para nós humanos identificar uma música como alegre ou triste é algo inato. Este processo é, no entanto, algo subjetivo, uma vez que um mesmo estímulo é por vezes identificado com emoções distintas por diferentes pessoas. Assim, o primeiro passo para qualquer sistema de MER é escolher uma taxonomia de emoção a utilizar no sistema, pois os resultados do processo de classificação serão representações das emoções segundo esta. As

várias taxonomias propostas na área da psicologia dividem-se em duas conceptualizações distintas.

2.3.1. Conceptualização Categórica da Emoção

A conceptualização categórica da emoção utiliza um conjunto de palavras, por exemplo tristeza ou felicidade, para reconhecer a emoção. Existem diversos modelos deste tipo, como por exemplo as emoções básicas propostas por Ekman (Ekman), que utilizam expressões faciais para categorização, como a Figura 2, e o círculo de adjectivos de Hevner [*Hevner's Adjective Circle*], criado por Kate Hevner (Hevner), que categoriza a emoção através de conjuntos de adjetivos divididos em vários grupos, como representado na Figura 3.

Figura 2 - Expressões Faciais usadas nas emoções básicas de Ekman[12]

[12] Figura 2.3, retirada da página 25, de (Panda R. E., 2019)

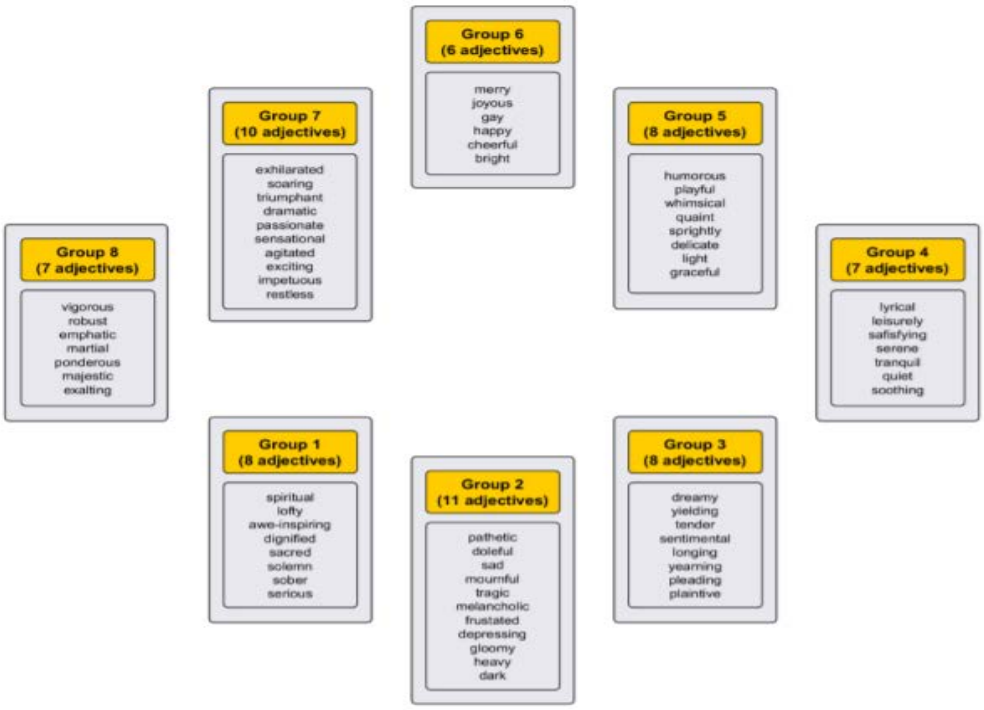

Figura 3 - Círculo de adjetivos de Hevner[13]

2.3.2. Conceptualização Dimensional da Emoção

A conceptualização dimensional da emoção aborda a emoção como "escalável e medível por descritores contínuos ou simplesmente por métricas multidimensionais" (E. Kim, et al., 2010), isto é, a categorização de emoção através de valores representados em planos dimensionais. Entre os vários modelos que existem, o mais utilizado é o modelo circumplexo de emoção de Russell [*Russell's Circumplex Model of Emotion*], representado na Figura 4, que utiliza quatro quadrantes, definidos pelos eixos de valência (*valence*) e pela excitação (a*rousal*), que representam no quadrante 1 as emoções felizes, no quadrante

[13] Figura 2.4, retirada da página 27, de (Panda R. E., 2019)

2 as emoções ansiosas ou agressivas, no quadrante 3 para as depressivas e no quadrante 4 as emoções calmas.

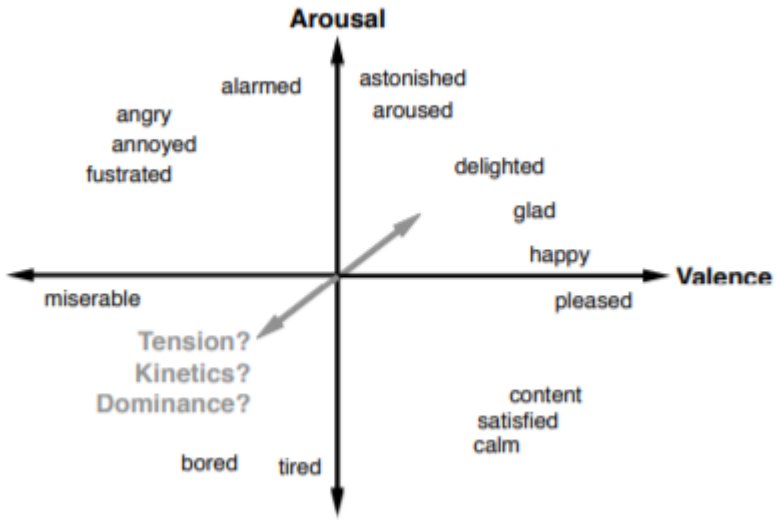

Figura 4 - Representação de emoções através do espaço *Valence-Arousal*, segundo o modelo de Russell[14]

2.3.3. Considerações Finais

Existem vários modelos em cada uma das duas abordagens, categóricas e dimensionais. As categóricas têm a vantagem de serem mais fáceis, próximos do que nós humanos estamos habituados a usar. Por outro lado, são mais limitados por dividirem um conjunto grande em N categorias, não permitindo por exemplo distinguir músicas dentro de uma categoria (100 músicas *tagged* como alegre são todas iguais para efeitos de pesquisa).

Os modelos dimensionais resolvem este problema, já que cada ponto num mapa 2 ou 3D representa uma emoção diferente. No entanto estes modelos levantam um novo conjunto de problemas: são computacionalmente mais complexos e também são mais complexos

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

para nós humanos, pois não estamos habituados a usar A e V para demonstrar a nossa emoção.

Assim, como o âmbito deste trabalho é criar um conceito de software, foi seguido o modelo categórico com os 4 quadrantes de Russell, e utilizado o *dataset* do conjunto de dados do paper: (Panda, Malheiro, & Paiva, Novel audio features for music emotion recognition, 2018).

2.4. Abordagem Típica dos Sistemas de MER

À semelhança de outros problemas de recuperação de informação (*information retrieval*), um sistema de reconhecimento de emoções em música contém por norma 3 partes distintas, tal como representado na Figura 5: construção de um conjunto de dados etiquetado emocionalmente (*dataset*), processamento e análise do sinal áudio e aprendizagem automática.

Na criação do *dataset*, um conjunto de exemplos musicais é selecionado e etiquetado emocionalmente por um conjunto de voluntários, seguindo a taxonomia selecionada. De seguida, os ficheiros áudio são processados através de algoritmos que irão fazer a extração e sumarização das suas características (por exemplo, o número médio de batidas por minuto), integrando os dados temporais (por exemplo em média e desvio-padrão) e preparando-os para descartar os possíveis valores que poderão reduzir a eficiência do modelo.

A fase de aprendizagem computacional pode ser dividida em 3 partes: treino, teste e classificação. O processo de treino é “uma representação matemática de um processo da vida real”[15], neste caso a identificação de padrões entre características do sinal e etiquetas emocionais que permitam a classificação de uma emoção, e é gerado na fase de treino

[15] https://medium.com/technology-nineleaps/some-key-machine-learning-definitions-b524eb6cb48

através do processamento de um algoritmo de aprendizagem sobre as características (ou *features*) de um *dataset*. Este processo é por norma realizado apenas com um subconjunto das músicas do *dataset* original, sendo as restantes reservadas para testar a precisão do classificador gerado.

A fase de testes é realizada depois de obtido um modelo treinado, onde as músicas do *dataset* que não foram utilizadas no treino do classificador são agora fornecidas ao modelo treinado a fim de obter a sua classificação emocional. Esta classificação prevista é comparada com a etiqueta real, obtendo um valor de precisão do classificador, o que nos permite por exemplo afirmar que o modelo é capaz de acertar em 60% das previsões.

Posteriormente, o processo de classificação em ambiente real passa pela obtenção de um novo sinal de áudio, extração das suas características (as mesmas utilizadas no processo de treino do modelo) e finalmente a classificação emocional utilizando o melhor modelo obtido no processo anterior. Esta classificação é uma classe resultante do processamento do modelo sobre as características extraídas do sinal.

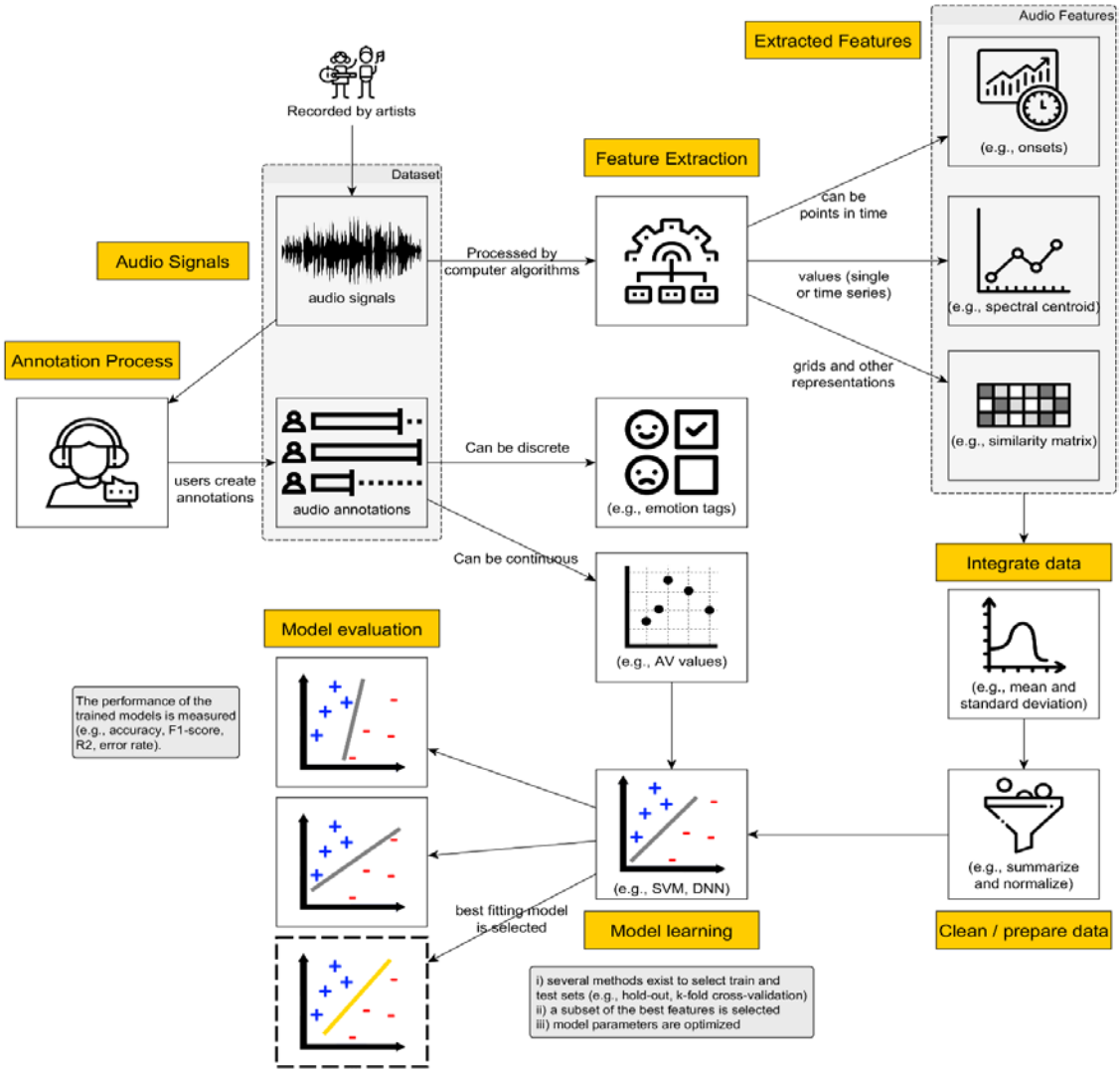

Figura 5 - Estratégia de aprendizagem computacional supervisionada aplicada em MER[16]

As secções seguintes apresentam detalhe adicional sobre cada uma das fases ilustradas.

2.4.1. Conjuntos de Dados ou *Datasets*

Um *dataset* é um conjunto de dados devidamente estruturados, balanceados e credíveis, composto pelo sinal de áudio e pela sua anotação, previamente dada por um humano, que irá ser utilizado com o intuito de treinar um modelo de classificação. Esta anotação pode

[16] Figura 3.3, retirada da página 102, de (Panda R. E., 2019)

ser obtida utilizando dados discretos (por exemplo etiquetas de emoção: triste, calma, entre outras) ou então valores contínuos (por exemplo os valores de excitação ("*arousal*") ou valência ("*valence*") de uma música. Apesar de parecer uma tarefa simples, é extremamente complicado encontrar um bom conjunto de dados para podermos utilizar na área de MER, uma vez que é uma tarefa complexa e intensiva em termos de recursos (voluntários), ainda mais considerando que nem sempre há um consenso acerca de emoções, como referido por (Chen, 2012). A maior parte dos projetos utilizam conjuntos de dados desenvolvidos pelos próprios colaboradores dos projetos, despendendo muito tempo para a realização de um conjunto de entradas válidas (no nosso caso músicas), resultando em conjuntos pequenos, não excedendo normalmente 1000 entradas (Chen, 2012). Tendo um conjunto de dados pequeno torna o processo de aprendizagem menos robusto e generalizável.

De entre os conjuntos que poderiam ser utilizados, a escolha recaiu sobre um *dataset* proposto por MOODetector (Panda, Malheiro, & Paiva, 2018), composto por 900 excertos musicais de 30 segundos, cada um etiquetado segundo uma taxonomia categórica, utilizando uma das "quatro classes emocionais pertencentes ao modelo *circumplex* de emoção de Russell [*Russell's Circumplex Model of Emotion*]. Este conjunto foi validado por um método semiautomático (anotações AllMusic[17], juntamente com uma validação humana mais simples), para reduzir os recursos necessários para construir um *dataset* completamente manual)" (Panda, Malheiro, & Paiva, 2018).

2.4.2. *Frameworks* para Tratamento do Sinal Áudio

Ao entrarmos no processo de recolha ou recuperação de informações de uma música, a extração de características de um sinal áudio é extremamente importante para todo o processo. As características são informações que são extraídas de um sinal áudio para que

[17] https://www.allmusic.com/

depois possam ser analisadas e se consiga obter um padrão, através de inteligência artificial (Moffat, Ronan, & D. Reiss, 2015). Estas características são a base de muitas pesquisas no campo do reconhecimento de emoção em música, mas também de outros campos de MIR como o de efeitos digitais de áudio (Panda R. E., 2019). Alguns exemplos simples destas características são o *zero crossing rate*, onde é verificado o número de vezes que o sinal passa pelo eixo dos x, ou por exemplo informação sobre o número de batidas por minuto. Pondo de parte todos os problemas acerca da definição e representação de uma emoção que aparecem na criação do *dataset*, também o processamento de sinal introduz novas dificuldades. Nomeadamente o facto de existir muita informação transmitida num sinal de áudio que ainda hoje é difícil de extrair utilizando um sistema computacional. Por outro lado, de entre as várias características, é difícil saber quais as melhores para identificar uma emoção.

Ao longo das décadas foram propostos vários algoritmos de processamento de sinal, que atualmente são agrupados em *frameworks* áudio. Algumas destas têm como principal objetivo serem mais intuitivas para utilizador (educacionais), enquanto outras foram desenvolvidas com o foco na eficiência computacional, por exemplo para serem utilizadas na indústria, ou são simples conceitos académicos (Panda R. E., 2019).

De seguida, serão apresentadas de forma breve algumas das *frameworks* que são mais vulgarmente utilizadas. Para uma explicação mais detalhada sobre estas consultar (Moffat, Ronan, & D. Reiss, 2015).

Essentia

Essentia é uma biblioteca *open-source* desenvolvida em C++ para análise de áudio e recolha de informação do sinal áudio. Implementa uma coleção de vários algoritmos, entre os quais as funções típicas, como filtros, para o processamento de sinal digital, caraterização estatística de dados e grande conjunto de descritores musicais espetrais, temporais, tonais e de alto nível. É multiplataforma e focada na robustez, velocidade

computacional e utilização de pouca memória. Esta biblioteca também permite a utilização de comandos e extensões em Python, o que leva ao desenvolvimento e obtenção de resultados muito rapidamente[18].

jMir

jMir é um conjunto de aplicações *open-source* implementadas em Java para a investigação na área de MIR. Pode ser utilizado para estudar música em diversos formatos, desde gravações áudio, codificações simbólicas como o MIDI e transcrições líricas, podendo também extrair (data mining) informações culturais da Internet. Inclui também ferramentas para gerir e criar perfis com base em largas listas de músicas e verificar áudio quanto a erros de produção. O jMir inclui também *software* para extrair características, aplicar algoritmos de aprendizagem computacional, aplicar verificadores de erros heurísticos e analisar meta-dados. O principal objetivo do jMir é providenciar *software* para investigação em classificação automática de música e análise de similaridade[19].

LibROSA

LibROSA é uma biblioteca desenvolvida em Python para análise de música e áudio que "fornece os componentes necessários para criar sistemas de MIR"[20]. Como referido pelos criadores desta biblioteca (McFee, et al., 2015), os principais objetivos desta são: ter uma curva de aprendizagem reduzida para investigadores mais familiarizados com MATLAB, garantir a padronização de interfaces, nomes de variáveis e configurações de parâmetros em várias funções de análise, manter compatibilidade com versões anteriores da biblioteca, assim como com implementações de referência existentes e ter um código legível,

[18] https://essentia.upf.edu/documentation/
[19] http://jmir.sourceforge.net/
[20] https://librosa.github.io/librosa/

documentação completa e testes exaustivos. Como a área de MIR está em constante evolução, é reconhecido que as implementações do LibROSA podem não representar o estado da arte a cada momento. Consequentemente, as funções são desenhadas para serem modulares, permitindo que os utilizadores se sirvam destas para implementar as suas próprias funções quando for necessário. Uma explicação mais detalhada está disponível em (McFee, et al., 2015).

Marsyas

Marsyas, acrónimo para *Music Analysis, Retrieval and Synthesis for Audio Signals* é uma *framework* de *software open-source* em C++ para processamento de áudio com ênfase específico nas aplicações MIR[21]. A *framework* oferece um conjunto de blocos básicos como filtros, transformadas, acumuladores, entre outros, que podem ser combinados em sequência ou paralelamente, que permitem implementar novos algoritmos de forma flexível e providenciar boa performance. Oferece também um conjunto de aplicações de demonstração das suas funcionalidades. Num entanto esta flexibilidade torna a *framework* mais complexa, não tendo documentação suficiente para mitigar este problema [22].

MIR Toolbox

MIR Toolbox é um conjunto integrado de funções (*toolbox*) escritas em MATLAB, dedicado à extração de características musicais como tonalidade, ritmo, timbre, melodia, entre outras, em sinais de áudio. O objetivo é oferecer uma implementação fácil de usar dos vários algoritmos computacionais na área de MIR. O *design* é baseado numa estrutura modular: os diferentes algoritmos estão decompostos em diferentes estágios, formalizados utilizando um pequeno conjunto de mecanismos elementares. Esses blocos de construção

[21] http://marsyas.info/
[22] http://marsyas.info/about/overview.html

formam o vocabulário básico desta ferramenta, que pode ser livremente articulado de diferentes maneiras novas e originais. Esses mecanismos elementares integram muitas vezes diferentes variantes propostas por abordagens alternativas de diferentes investigadores, que os utilizadores podem selecionar e parametrizar. Esta ferramenta foi inicialmente desenvolvida no contexto do projeto Brain Tuning[23], financiado pela União Europeia. O objetivo principal foi investigar a relação entre as características musicais e emoção induzida pela música e atividade neuronal apresentada"[24]. O facto de ser escrita em MATLAB e mais virada para a investigação torna-a uma das *frameworks* mais pesadas em termos computacionais, não tendo o desempenho equivalente às alternativas em C++.

Considerações Finais

Existem hoje diversas *frameworks* disponíveis para extrair características do sinal áudio. Essas características podem ser, por exemplo, de baixo nível, isto é, aquelas que estão mais perto do sinal áudio ou por exemplo alto nível, que estão mais longe do sinal áudio, estando por isso mais perto do que nós humanos percebemos e retiramos de uma música. Estas *frameworks* têm sido usadas em diversos estudos ao longo das últimas décadas, estando as principais características destas resumidas na Tabela 1**.** Com base nos resultados decidimos escolher a *framework* Essentia para este projeto, devido à grande panóplia funcionalidades principais que fornece. Além disso tem vantagens em termos de desempenho, devido ser desenvolvida em C++, é *open-source*, pode ser invocada utilizando Python e não é necessário MATLAB para utilizar. Para informações mais detalhadas acerca das comparações destas *frameworks*, consultar (Moffat, Ronan, & D. Reiss, 2015).

[23] http://braintuning.fi/
[24] https://www.jyu.fi/hytk/fi/laitokset/mutku/en/research/materials/mirtoolbox

	Essentia	jMir	LibROSA	Marsyas	MIR Toolbox
Características de alto nível	Sim	Não	Sim	Sim	Sim
Características de baixo nível	Sim	Sim	Sim	Sim	Sim
Filtros	Sim	Não	Não	Não	Sim
Reamostragem	Sim	Sim	Sim	Sim	Sim
APIs	C/C++ Python R	Java	Python	C/C++ Python Java	MATLAB

Tabela 1 - Comparação das diferentes ferramentas de extração de características[25]

2.4.3. Algoritmos de Aprendizagem Computacional

Depois da recolha de informações do sinal áudio, a continuação do projeto passa por dar inteligência ao sistema. A inteligência artificial tem vindo cada vez mais a ser utilizada no nosso dia-a-dia nas mais diversas áreas. Aprendizagem computacional é uma área de inteligência artificial que tem como objetivo a "fornecer a um sistema a habilidade de automaticamente aprender e melhorar a sua experiência sem ser explicitamente

[25] Tabela baseada em tabela 1, retirado da página DAFX-4, de (Moffat, Ronan, & D. Reiss, 2015)

programado" (Expert System, s.d.), através de aprendizagem supervisionada (existência de dados etiquetados) ou não supervisionada. Este trabalho segue a abordagem de classificação proposta em (Panda, Malheiro, & Paiva, Novel audio features for music emotion recognition, 2018), que tem por base a aprendizagem supervisionada, isto é, neste tipo de aprendizagem os dados de entrada para treinar o modelo de classificação contêm etiquetas ou classes (Tabela 2), ao contrário da linguagem não supervisionada (B. Kotsiantis, 2007).

Característica 1	**Característica 2**	**….**	**Característica N**	**Class**
xxxx	x		xx	Feliz
xxxx	x		xx	Feliz
xxxx	x		xx	Triste
….	….		….	….

Tabela 2 - Instâncias com dados de entrada e suas correspondentes saídas[26]

Dentro da aprendizagem computacional supervisionada existem vários algoritmos que são abordados de seguida. Uns baseados na lógica, como Árvores de Decisão, outros baseados em aprendizagem estatística, como os k Vizinhos Mais Próximos (kNN, *k-Nearest Neighbour*) ou as Máquinas de Vetores de Suporte (SVM, *Support Vector Machines*).

[26] Baseada na Tabela 1, da página 249, de (B. Kotsiantis, 2007)

Árvores de Decisão

Árvores de Decisão, exemplificado na Figura 6, são "árvores que classificam instâncias fazendo uma organização baseada no valor das características" (B. Kotsiantis, 2007), isto é, um método baseado em tomadas de decisão. O seu funcionamento passa pela existência de nós conectados por ramos, em que cada nó representa uma característica e os ramos são os seus possíveis valores. Utilizando como exemplo a Figura 6, a condição começa no nó raiz, *at1* e existem três caminhos possíveis, os ramos *a1*, *b2* e *c1*. A partir do nó raiz vai-se tomando uma decisão consoante o valor dos ramos. Caso na característica *at1* se verifique o ramo (valor) *a1*, então a decisão continua para a característica *at2*. Se por outro lado a característica *at1* verifique as condições *b1* ou *c1* o resultado de classificação será "Não".

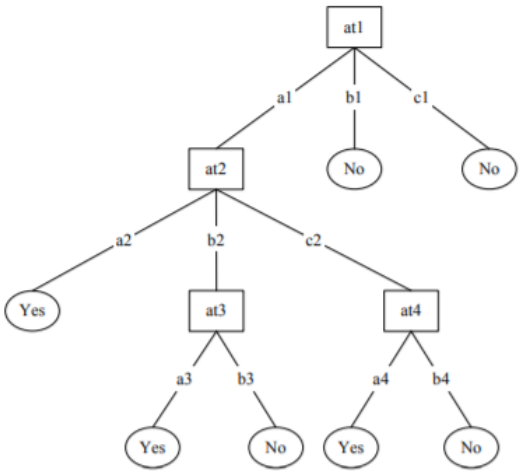

Figura 6 – Exemplo de uma árvore de decisão[27]

[27] Figura 2, retirada da página 251, de (B. Kotsiantis, 2007)

k-Vizinhos Mais Próximos (KNN)

k-Nearest Neighbour (kNN), exemplificado na Figura 7, é "baseado no princípio de que as instâncias de um conjunto de dados geralmente existirão muito próximas a outras instâncias que possuem propriedades semelhantes" (B. Kotsiantis, 2007). Este algoritmo utiliza um conjunto de eixos que representam as características, por exemplo o eixo dos X poderia ser batidas por minuto e dos Y a duração do som. Neste referencial de N dimensões são colocados os casos de treino. A classificação de um novo exemplo é dada através da sua colocação no espaço, sendo-lhe atribuída uma etiqueta igual à que estiver em maioria entre os seus k vizinhos mais próximos. Por exemplo, na Figura 7, a instância verde é o ponto sem etiqueta, que tem uma certa posição em relação a uns eixos. Existem instâncias de cor azul e vermelhas que já foram etiquetadas. Se o número de vizinhos (k) for 3, a etiqueta resultante da classificação da instância verde vai ser a mesma que as instâncias vermelhas, caso seja 5, a etiqueta vai ser a mesma das instâncias azuis.

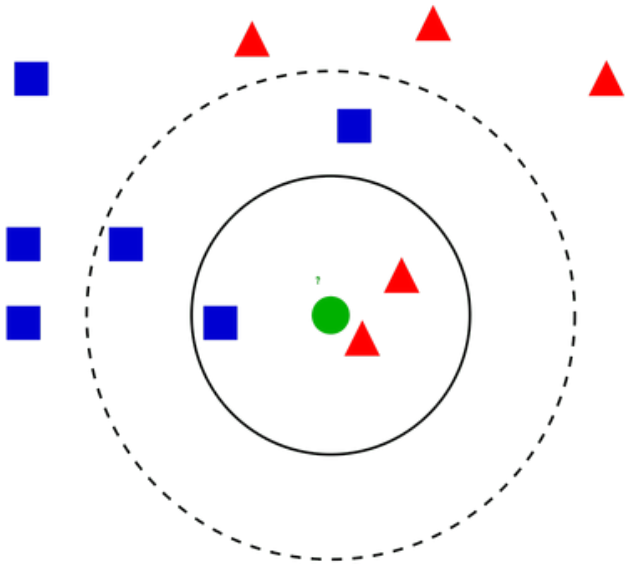

Figura 7 - k-Nearest Neightbour[28]

[28] Retirado de https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

Máquinas de Vectores de Suporte (SVM)

Máquinas Vetores de Suporte (SVM), representado na Figura 8, são um dos algoritmos de aprendizagem mais utilizados atualmente. Baseia-se na "noção de "margem" - um dos lados do híper-plano que separa duas classes de dados" (B. Kotsiantis, 2007), isto é, o algoritmo tenta encontrar uma linha, denominada híper-plano, que consiga separar o mais possível as classes através da maximização das margens (B. Kotsiantis, 2007). Para melhor compreensão do algoritmo ver a secção 7, na página 260 de (B. Kotsiantis, 2007).

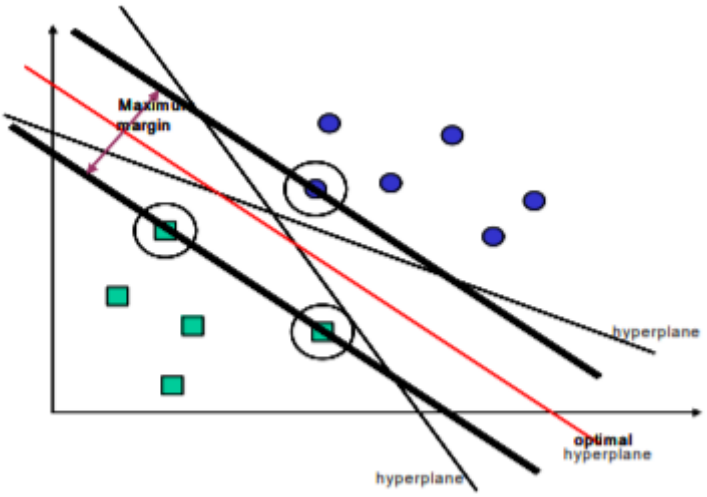

Figura 8 - SVM, processo de maximização da margem[29]

Considerações Finais

Como se pode verificar na Tabela 3, os algoritmos kNN e SVM são dos que obtêm melhores resultados. Isto deve-se à precisão que cada um deles tem e à tolerância demonstrada a valores considerados "*outliers*", isto é, valores que podem trazer imprecisão ao sistema quer por serem valores atípicos, quer por redundância de valores. Porém, como demonstrado em (B. Kotsiantis, 2007), o algoritmo kNN demonstra algumas lacunas quer

[29] Figura 9, retirada da página 261, de (B. Kotsiantis, 2007)

termos de espaço necessário para o armazenamento necessário para a classificação, quer em termos de processamento necessário para tentar obter um bom valor de k. Devido a isso e suportado também pela opção tida em (Panda, Malheiro, & Paiva, Novel audio features for music emotion recognition, 2018) a escolha irá recair sobre o algoritmo SVM. Existia também a opção da utilização de algoritmos de aprendizagem profunda (*deep learning)*, porém estes algoritmos têm como base enormes conjuntos de dados anotados e estes tipos de *datasets* não existem atualmente, pelo que a utilização destes algoritmos só poderá ser considerada futuramente. Para melhor compreensão sobre o funcionamento dos algoritmos e sobre as suas vantagens e desvantagens, consultar (B. Kotsiantis, 2007).

	Árvores de Decisão	**kNN**	**SVM**	**Neural Networks**
Precisão	**	**	****	***
Velocidade de classificação	****	*	****	****
Tolerância a valores em falta	***	*	**	*
Tolerância a atributos irrelevantes	***	**	****	*
Tolerância a atributos redundantes	**	**	***	**
Tolerância a ruído	**	*	**	**

Tabela 3 - Comparação dos diferentes algoritmos de classificação supervisionada[30]

[30] Baseada da Tabela 4, da página 263, de (B. Kotsiantis, 2007)

2.4.4. Implementações de Máquinas de Vector de Suporte (SVM)

Uma vez escolhido o algoritmo é necessário escolher de uma biblioteca que forneça a sua implementação. Surgem então duas grandes referências a LIBSVM e Scikit-Learn que serão abordadas a seguir.

LIBSVM

LIBSVM é um *software* integrado para classificação, regressão e estimação da distribuição [*support vector classification, regression* and *distribution estimation*]. Suporta também classificação multi-classe. LIBSVM fornece uma interface de programação de aplicações simples, que os programadores podem facilmente utilizar nos seus próprios programas[31]. O código fonte está possível em C++ e Java, e a biblioteca tem extensões para Python, R, MATLAB, Perl, Ruby, Weka, Common LISP, entre outras.

Scikit-Learn

Scikit-Learn é um módulo em Python que integra um conjunto vasto de algoritmos de aprendizagem computacional que são parte do estado da arte para problemas classificação, supervisionada ou não, média escala. Este módulo pretende trazer a aprendizagem computacional aos não especialistas, utilizando para isso uma linguagem de alto nível (Python). A ênfase do projeto é posta na facilidade de uso, performance, documentação e consistência da API. Exige poucas dependências de outro *software* e é distribuída com a

[31] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

licença BSD para que possa ser utilizada tanto por estudantes académicos como fins comerciais”[32].

Considerações Finais

Ambos os *softwares* que fazem implementação de SVM são extremamente completos, fornecendo muitas funcionalidades para a implementação do algoritmo SVM. Porém, a biblioteca Scikit-Learn incorpora a biblioteca em C++ do LibSVM e, ao contrário das outras bibliotecas, é compilada, o que permite que tenha mais eficiência. Além disso, os *bindings* para Python do software Scikit-Learn têm menos 40% de *overhead* que os *bindings* originais do SVM (Pedregosa, et al.).

[32] http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

Capítulo 3
Planeamento e Desenvolvimento

A solução proposta passa pelo desenvolvimento de um sistema de reconhecimento emocional em música áudio organizado seguindo uma arquitetura de microsserviços, com a comunicação entre estes a ser realizada através de um serviço de filas de mensagens, que são consumidas também por uma aplicação Web. Neste capítulo, é abordado o planeamento da solução final, as tecnologias consideradas e o seu processo de desenvolvimento. Na secção 3.1 é abordada a arquitetura utilizada para o desenvolvimento do projeto. De seguida, na secção 3.2, são abordadas as diferentes soluções existentes para a orquestração de *containers.* Na secção 3.3 são abordados os mecanismos de comunicação entre processos. Para terminar este capítulo, a secção 3.4 apresenta as linguagens de programação que foram consideradas para a implementação do sistema.

3.1. Arquitetura Geral do Sistema

Como abordado anteriormente, um dos problemas da área de MER passa por haver poucas aplicações reais que tenham como objetivo principal de fazer tratamento do áudio e utilizá-lo para obter uma classificação da emoção presente neste. Entre as possíveis razões para isto, estão a complexidade do tema e o facto de ser ainda hoje um problema em aberto, onde novos avanços vão sendo propostos, mas ainda longe de produzirem uma solução robusta para situações reais. Torna-se por isso necessário desenvolver uma solução que considere estes problemas e que seja fácil de se adaptar a eventuais mudanças. Parte deste objetivo é conseguido através da utilização de certos conceitos de engenharia de *software* e metodologias de desenvolvimento *Agile*. Pressupõe-se o desenvolvimento de uma solução robusta, escalável e capaz de suportar todo o processo pesado de tratamento de sinal, como extração de características e extração do áudio, sem que interfira com o restante funcionamento da aplicação. Para cumprir estes requisitos obtendo o melhor

resultado possível, o primeiro passo necessário consiste em identificar qual a melhor arquitetura a escolher para o projeto.

A definição de arquitetura geral da solução consiste numa "descrição do sistema que ajuda a entender como o sistema se comportará"[33]. As várias tarefas do sistema a desenvolver têm requisitos de tempo e processamento bastante díspares, pois têm vários passos complexos e distintos, uns mais demorados que outros. Por exemplo, o tempo que demora a extrair características de um sinal áudio é bastante maior que o necessário para obter a classificação tendo um modelo já treinado. Desta forma, um cenário ideal seria ter estas tarefas divididas em blocos isolados e independentes, cada um responsável exclusivamente pela sua tarefa. Como ilustrado pela Figura 9, é necessário que exista um elemento central, por exemplo uma fila de mensagens, com o objetivo de permitir a comunicação entre a aplicação Web e os vários componentes que fazem o tratamento do sinal áudio. Assim, existem duas abordagens que foram tomadas em conta: a arquitetura tradicional (monolítica), caracterizada pelo desenvolvimento num só bloco, ou a arquitetura de microsserviços, caracterizada pelo desenvolvimento de vários blocos.

[33] https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21328

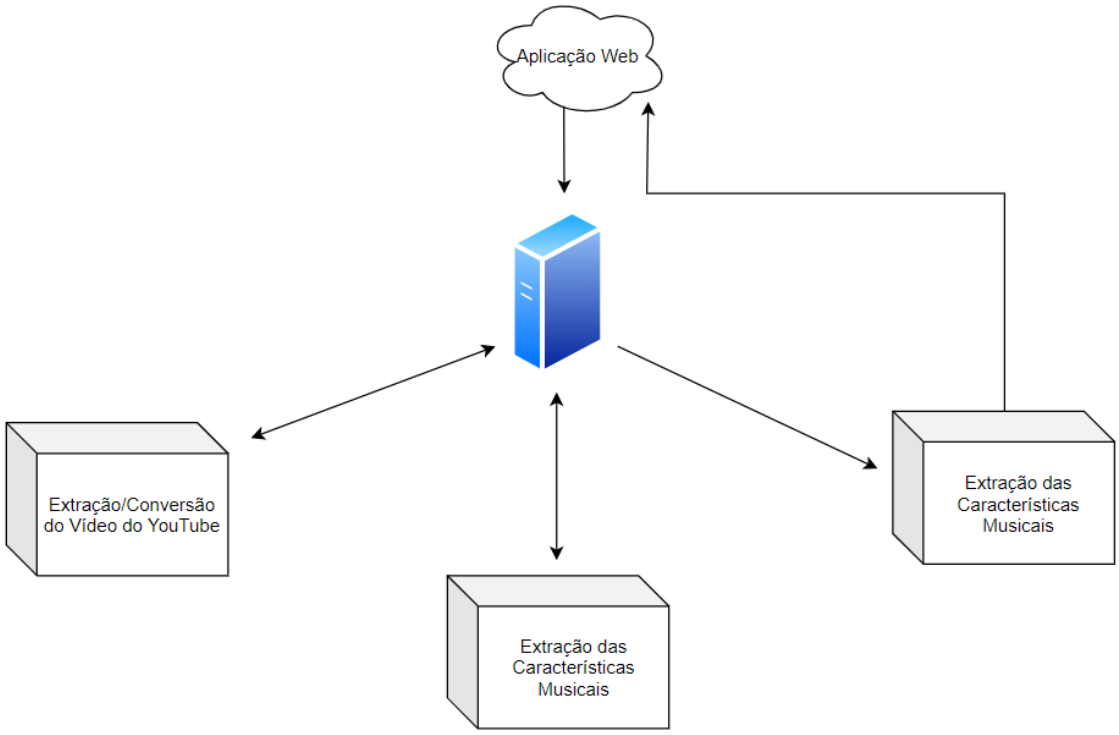

Figura 9 - Arquitetura Geral da Solução

A arquitetura tradicional, ou monolítica, é a abordagem típica presente, hoje em dia, em sistemas comuns e caracteriza-se por "uma aplicação com uma única base de código/repositório, por normal grande, que oferece dezenas ou centenas de funcionalidades usando interfaces diferentes como páginas HTML, serviços Web e/ou serviços REST" (Villamizar, Castro, Verano Merino, & Casallas, 2015). Isto é, como referido na Figura 10, todos os diferentes componentes pertencem uma única aplicação sob uma única plataforma. Esta abordagem está vastamente disseminada pela indústria, mas com o surgimento de computação para a nuvem esta arquitetura tem vindo a ser substituída pela utilização de microsserviços, pois a sua manutenção e gestão torna-se dispendiosa, especialmente quando se trata de aplicações complexas com as características anteriormente referidas. A arquitetura tradicional tem, no entanto, vários pontos a favor, como por exemplo a facilidade de desenvolvimento, em especial no início do projeto, facilidade em testar o sistema e facilidade em colocar o trabalho desenvolvido em

produção (Haq, 2018). Os problemas principais aparecem mais tarde, com o crescimento da aplicação, na dificuldade de manutenção e deteção/correção de erros e na sua escalabilidade (Haq, 2018). Nesta arquitetura, cada vez que é adicionada uma nova funcionalidade ou é necessária uma atualização, torna-se necessário voltar a fazer a implementação (*deploy*) da aplicação completa novamente para o servidor, carregando todos os módulos e bibliotecas que lhe pertencem (Haq, 2018). Este processo por norma demora algum tempo, o que pode levar a que a aplicação tenha disrupções (seja colocada *offline*) durante esse tempo. Nesta arquitetura, também quando há congestionamento ou algum erro, a execução da aplicação é interrompida ou torna-se mais lenta, levando a que o sistema esteja parado algum tempo. A manutenção da aplicação e deteção de erros torna-se um processo cada vez mais complexo, que aumenta com a quantidade de funcionalidades que vão surgindo à medida que a aplicação vai crescendo (Haq, 2018). Com o crescimento da aplicação, também os recursos utilizados serão cada vez mais, levando a que seja necessário escalar a aplicação. Escalar uma aplicação seguindo uma arquitetura tradicional é por norma uma tarefa difícil, sendo a abordagem tipicamente escolhida a de escalar horizontalmente (Haq, 2018), isto é, adicionando mais máquinas com a aplicação (réplicas) e utilizando um balanceador de carga para fazer a gestão de pedidos. Existem situações em que tal não é possível, sendo necessário escalar a aplicação verticalmente, isto é, dando mais recursos à máquina onde está alojada a aplicação. No caso deste projeto, o desenvolvimento seria de uma aplicação sobre um só bloco que seria responsável pela transferência de vídeos, extração e conversão para áudio, recolha de características e classificação. Estes processos são algo pesados do ponto de vista computacional, pelo que cada vez que o sistema os estivesse a executar resultaria em tempos de espera. Como já referido, uma solução para este problema passa pela utilização de várias máquinas correndo a aplicação e de um balanceador de carga que distribuía novos pedidos pelas várias máquinas disponíveis, diminuindo o tempo de espera e mantendo a aplicação operacional. Tal abordagem apresenta diversas desvantagens, sendo a maior o custo associado. Existia também a opção da utilização de *threads* para todas as tarefas de processamento, distribuindo (paralelizando) o processamento das tarefas numa mesma máquina pelos vários núcleos de processamento e garantindo que a aplicação continua a responder. No entanto, a gestão de *threads* é um processo complexo e quando um

problema causa a falha na parte do processamento, por norma toda a aplicação falha. Existem ainda todos os outros problemas anteriormente referidos, como a complexidade no tratamento de erros, manutenção ou implementação que contribuem para aumentar a dificuldade desta opção. Devido a isso, torna-se necessária a escolha de uma outra arquitetura que permita repartir toda a aplicação em pequenos serviços independentes e isolados, responsáveis pela execução das tarefas distintas como a extração da música e conversão para áudio, recolha de características e classificação.

A arquitetura de microsserviços é uma arquitetura "inspirada pela computação orientada a serviços que tem vindo a ganhar popularidade" (Dragoni, et al.). A ideia da utilização desta arquitetura, ilustrada na Figura 10, passa pela criação de pequenos serviços independentes que comunicam através de mensagens. Normalmente, a implementação destes microsserviços é realizada sobre *containers* (pacote de software leve e autónomo que inclui tudo o que é necessário para correr uma aplicação) e são orquestrados através de ferramentas como o Docker[34] ou Kubernetes[35], abordados mais á frente no capítulo 3.2.

Uma vez que a computação para nuvem se tem tornado uma das maneiras mais viáveis de disponibilizar aplicações e de utilizar *containers*, várias empresas têm vindo a modificar o seu paradigma no desenvolvimento mudando de arquiteturas monolíticas para arquitetura de microsserviços. A arquitetura de microsserviços fornece grandes benefícios em termos de desenvolvimento contínuo e manutenção de grandes aplicações, uma vez que é mais fácil incluir novas mudanças, substituindo os módulos de um sistema (*containers*) se estes forem independentes, sem ter de tornar o sistema indisponível. A arquitetura apresenta diversas outras vantagens como a facilidade de aplicação de testes devido aos serviços serem pequenos (embora possa ser mais complicado testar a aplicação de forma automatizada como um todo), facilidade de implementação por serem independentes, torna mais fácil isolar erros, isto é, caso um serviço falhe não interfere com a execução dos outros serviços, permite a atualização ou modificação de casa serviço sem comprometer o resto dos serviços. Acrescenta flexibilidade na escolha de tecnologias utilizadas em cada

[34] https://www.docker.com/
[35] https://kubernetes.io/pt/

serviço, permitindo escolher a melhor solução a ser utilizada para um dado objetivo/tarefa. Isto é, permite que, por exemplo, o desenvolvimento de um serviço seja feito em JavaScript correndo sobre Node.js e outro utilize Python ou outra linguagem qualquer, uma vez que são independentes e comunicam através de mensagens. Esta abordagem é ainda fácil escalar tanto horizontalmente (mais nós) como verticalmente (mais recursos em cada nó) (Haq, 2018). "Escalar aplicações monolíticas é sempre mais difícil do que escalar microsserviços porque numa é necessário escalar toda a aplicação e fazer a implementação de todo o código em vez de dimensionar a parte da aplicação que exige mais recursos" (Kalske, Makitalo, & Mikkonen, 2018). Resumindo, ao escalar uma aplicação monolítica fornecemos mais recursos (verticalmente) ou duplicamos a funcionalidade (horizontalmente) de toda aplicação, mesmo que seja só uma tarefa específica da aplicação a necessitar de mais processamento. Ao passo que numa solução de microsserviços são dados mais recursos apenas ao serviço específico que esteja a necessitar. Porém, a arquitetura de microsserviços também tem alguns problemas. Alguns desses problemas devem-se ao facto de se lidar com a complexidade de criar um sistema distribuído. Apesar de ser fácil testar os serviços independentes, torna-se mais difícil testar a funcionalidade dos mesmo como um todo em relação a uma aplicação monolítica. São necessários mecanismos de comunicação entre serviços e, uma vez que estamos a executar vários serviços distintos sobre vários *containers*, que por sua vez executam sobre máquinas (físicas ou virtuais), o consumo de memória aumenta (Haq, 2018).

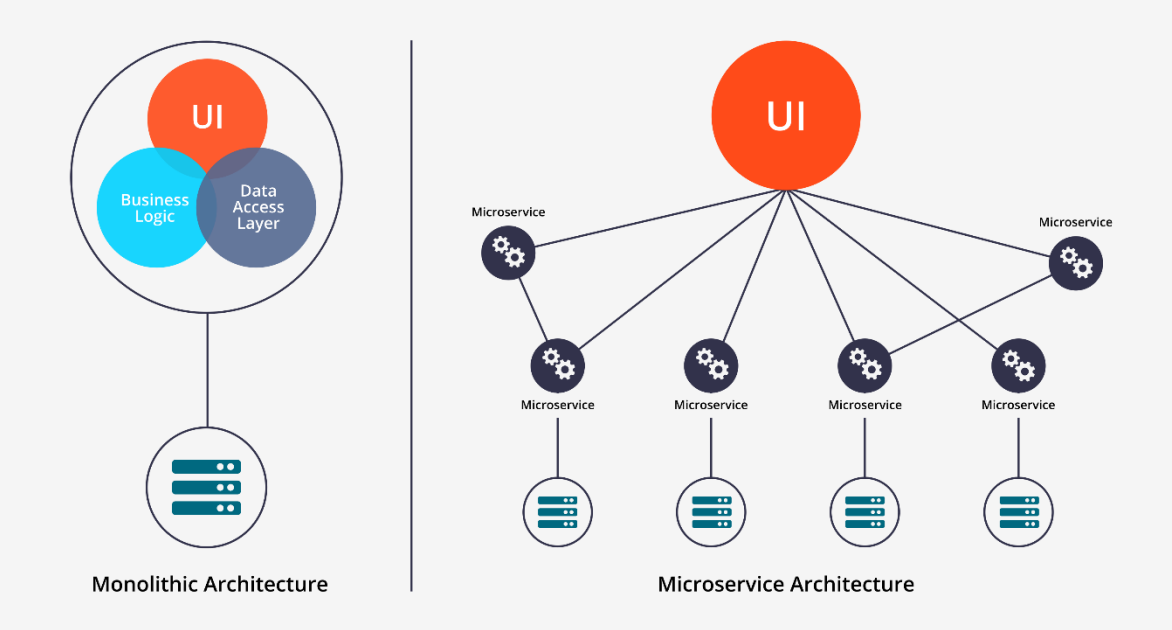

Figura 10 - Arquitetura Monolítica e Arquitetura de Microsserviços[36]

Concluindo, ambas as arquiteturas apresentadas têm os seus prós e contras. No entanto, 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. Caso seja necessário alterar algum dos serviços do sistema, por exemplo para adicionar uma nova tecnologia, atualizar bibliotecas ou alterar a própria abordagem implementada, basta preparar e atualizar o serviço em causa, ficando as suas tarefas em espera na fila de mensagens ou resolvidas por outro nó caso haja replicação. Por outro lado, caso haja erros ou congestionamento nos serviços, o normal funcionamento da aplicação não irá ser tão comprometido, bastando lançar novos serviços, algo difícil de obter com arquitetura monolítica.

[36] Figura retirada de https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4

3.2. Mecanismos de Orquestração de *Containers*

Para a implementação do sistema usando microsserviços é necessária a utilização de um mecanismo que permita a execução de *containers*. Os *containers* (contentores) são uma "unidade padrão de *software* que permite o encapsulamento de uma aplicação, incluindo o seu código e todas as dependências deste, para que seja utilizada de maneira rápida e confiável em vários ambientes de computação diferentes"[37]. A utilização deste tipo de soluções tem vindo a aumentar devido à sua flexibilidade, baixo peso computacional, facilidade na sua manutenção e mudança ao longo do tempo (mutação), portabilidade e associação a outros *containers*.

Existem hoje dois principais mecanismos que permitem a gestão e orquestração de *containers* – Docker e Kubernetes. O mecanismo escolhido para o desenvolvimento deste projeto foi o Docker pelo estado de maturidade, pelas funcionalidades e pela sua extensa documentação disponível.

3.2.1. Docker

O Docker é uma plataforma que permite desenvolver e executar aplicações, independentemente do ambiente computacional, utilizando para isso *containers*. Estes *containers* são independentes e a plataforma fornece diversas políticas de configuração, tais como a opção de reinício caso um *container* falhe ou até em caso de falha do próprio Docker. Como representado na Figura 11, o Docker assenta sobre o sistema operativo da máquina e consiste em *docker engine*, um mecanismo para a criação e gestão de *containers* e *docker hub*[38], um repositório para gerir e partilhar imagens dos *containers*. No caso do Docker engine, este pode ser executado em modo normal ou em enxame (*swarm*).

[37] https://www.docker.com/resources/what-container

[38] https://docs.docker.com/engine/

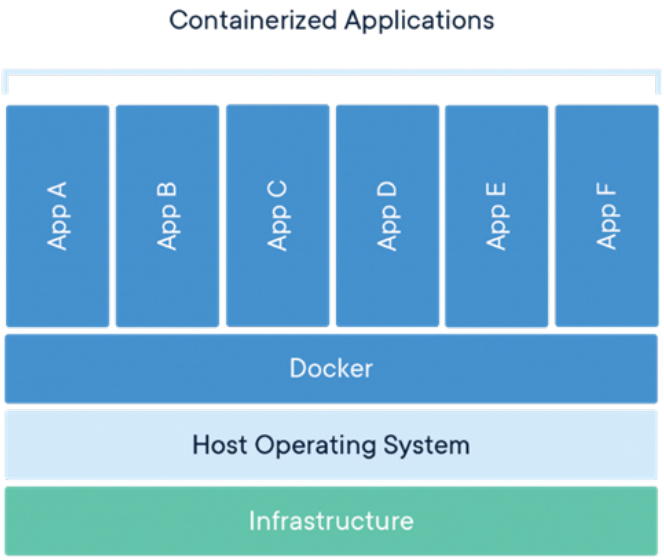

Figura 11 - Aplicações executadas em *containers* de Docker[39]

O *docker engine* é o mecanismo responsável pela criação e gestão de *containers* e que permite que eles sejam executados em qualquer ambiente de maneira consistente, fazendo *build* das imagens e depois *push* para um determinado repositório de imagens. Está disponível para vários sistemas operativos como CentOS, Debian, Fedora, Oracle Linux, Ubuntu, Windows, entre outros[40].

As aplicações desenvolvidas e "empacotadas" são chamadas de *docker images*, "pacotes de *software* leves, autónomos e executáveis que contém tudo o que é necessário para a aplicação que contém correr: código, ferramentas, bibliotecas e configurações"[41]. Estas imagens tornam-se *containers* quando estão em execução no *Docker Engine*.

[39] Figura retirada de https://www.docker.com/resources/what-container
[40] https://www.docker.com/products/docker-engine
[41] https://docs.docker.com/get-started/#docker-concepts

O *docker hub* é um repositório central de imagens que permite aos utilizadores disponibilizar (de forma pública ou privada) os seus *containers*, permitindo que depois sejam descarregadas e reutilizadas por outros utilizadores. O *docker hub* oferece um recurso centralizado que "permite a procura, distribuição e gestão de imagens dos *containers*, colaboração singular ou em equipa e automação de fluxos de trabalho em todo o pipeline de desenvolvimento"[42].

O *Dockerfile* é um ficheiro de texto onde se define a configuração deste. Nomeadamente o nome da imagem base a utilizar, o que é instalado dentro do *container* e comandos a executar. O acesso a recursos como interfaces de rede e *drivers* do disco é virtualizado dentro deste ambiente, isolando-o do resto sistema, sendo por isso necessário mapear as portas entre o interior do *container* e o mundo externo e especificar os arquivos que se pretendem copiar para esse ambiente. Depois de definido este ficheiro, a compilação da aplicação irá ser executada sempre de maneira semelhante, seguindo as instruções que este indica[43]. Isto é, o *Dockerfile* constrói uma imagem, contendo a definição do ambiente onde a aplicação vai correr.

O *Docker Compose* é a "ferramenta quer permite definir e executar uma aplicação que está dividida em vários *containers* de Docker, através de um ficheiro no formato YAML (YAML Ain't Markup Language – formato de codificação de dados)"[44]. Depois de definir a configuração necessária[45], com um único comando torna-se possível dar início a todos os serviços que estão definidos. Nesse ficheiro pode-se fazer referência a imagens presentes no *Docker Hub*, fazer referência a um ou vários ficheiros *Dockerfile* e até indicar as políticas de reinício para quando houver erros em *containers* ou o *Docker Engine* falhar.

O Docker fornece também uma ferramenta que permite fazer o provisionamento e gestão de *hosts*, a *docker machine*. Esta ferramenta permite "iniciar, inspecionar, parar e reiniciar um *host* sob gestão, atualizar o cliente e o *daemon* do Docker e configurar um cliente para

[42] https://docs.docker.com/docker-hub/
[43] https://www.docker.com/resources/what-container
[44] https://docs.docker.com/compose/
[45] A configuração do Docker Compose é efectuada no ficheiro *docker-compose.yml*

falar com outros *hosts"*[46]. Além da *docker machine* o Docker tem ainda um orquestrador de criação e gestão de *clusters* nativo, o *Docker Swarm*. É através deste orquestrador que é possível ter o Docker em modo de enxame. Um enxame (*swarm*) é um conjunto de máquinas distintas que estão a executar Docker e que entraram num *cluster*. Depois disso acontecer, os comandos normalmente executados num dos *hosts* passam a ser distribuídos e executados no *cluster*, através do *swarm manager*. As máquinas presentes no *cluster* podem ser virtuais ou reais e são referidas como *nodes*, sendo os *containers* distribuídos por estas Este modo permite utilizar estratégias para que as máquinas mais utilizadas sejam menos sobrecarregadas que as menos utilizadas, na altura de criação de *containers*. Caso um dos *containers* deixe de responder, o *swarm manager* lança automaticamente novos *containers*. Para este projeto, foram realizados testes de criação de *containers* e imagens para compreensão do *docker swarm* através da criação de várias máquinas virtuais com Docker.

3.2.2. Kubernetes

Kubernetes é um sistema de orquestração de *containers* desenvolvido inicialmente pela Google. Introduziu diversas funcionalidades antes de existirem em Docker, como por exemplo a facilidade no *deploy*, o *clustering* e especialmente o suporte para o balanceamento. A sua utilização é um pouco mais complexa do que o Docker e está normalmente associada ao Google Cloud. Tanto o Docker como o Kubernetes estão em rápido crescimento e o Kubernetes fornece a maior panóplia ferramentas de orquestração de *containers* atualmente, porém a tecnologia escolhida foi o Docker, pois a curva de aprendizagem inicial é mais simples que a curva de aprendizagem do Kubernetes e já havia alguma experiência na utilização da mesma.

[46] https://docs.docker.com/v17.09/machine/overview/#what-is-docker-machine

3.3. Mecanismos de Comunicação entre Processos

Como referido anteriormente, ao utilizar uma arquitetura de microsserviços são criados pequenos serviços independentes. Dada essa independência é necessário haver um mecanismo que permita a comunicação entre esses serviços. Estes mecanismos são denominados de *message brokers* e a sua função é a de traduzir uma mensagem para um protocolo formal de mensagens do emissor, para o protocolo formal de mensagens do recetor[47]. Isto é, a criação de um serviço centralizado de armazenamento de mensagens, por exemplo um servidor, onde seja possível a outros serviços produzir e consumir as mensagens armazenadas. As comunicações nestes mecanismos são, normalmente, assíncronas devido às comunicações síncronas serem comunicações em que é necessário que dois pontos que pretendem comunicar estejam sincronizados, normalmente através de um canal mútuo aos dois. Ao fornecer mecanismos de comunicação assíncrona e de armazenamento de mensagens, estas plataformas permitem que várias aplicações produzam e consumam mensagens sem que seja necessário haver sincronização entre estas. Isto leva a que seja ainda possível distribuir as mensagens por vários consumidores e elimina o tempo e recursos normalmente despendidos na sincronização entre as várias aplicações.

Existem dois modelos em que os mecanismos de comunicação podem operar: o modo de *queuing* e o modo *publish/subscribe*. No modo de *queuing*, todos os consumidores podem ler do *server* e cada registo vai para um desses consumidores. No modo *publish/subscribe* existe um canal onde cada vez que existe um novo registo este é entregue a todos os subscritores desse canal. Cada um destes tem uma vantagem e uma fraqueza, no modo *queuing*, a vantagem é o balanceamento de mensagens pelos vários consumidores, a fraqueza é que uma vez consumida, a mensagem sai do tópico. No modo *publish/subscribe* tem a vantagem de partilhar a mensagem por todos os subscritores, mas não permite escalar[48]. Existem várias soluções que utilizam esses

[47] https://medium.com/@xaviergeerinck/an-introduction-to-message-brokers-9bd203b4ebbd
[48] https://kafka.apache.org/intro.html

mecanismos como o Apache Kafka, Open Message Queue, RabbitMQ, entre vários outros. O *software* escolhido para este projeto foi o RabbitMQ pelas razões descritas em 3.3.4. A seguir são apresentados alguns destes.

3.3.1. Apache Kafka

Apache Kafka é uma plataforma *open-source* de transmissão distribuída, escrita em Java e Scala, que fornece três funcionalidades semelhantes a uma fila de mensagem, com mecanismos de *publish/subscribe* de recursos, guarda transmissão de registos de maneira contínua e tolerante a falhas e processa a transmissão dos registos conforme eles vão ocorrendo. Esses registos são guardados em tópicos, que por sua vez estão divididos em partições. Como ilustrado Figura 12, esta plataforma é composta por quatro APIs principais:

- *Producer API*, que permite publicar registos nos tópicos
- *Consumer API*, que permite consumir/aceitar esses registos guardados nos tópicos para depois serem processados
- *Streams API*, que transforma transmissões de *input* para transmissões de *output*
- *Conector API*, que permite a utilização de produtores e consumidores reutilizáveis, conectando um tópico a uma aplicação.

A comunicação entre o servidor e os clientes é feita com recurso a um protocolo binário sobre o protocolo TCP, que define as trocas de mensagens entre cliente e servidor como pares de mensagens pedido/resposta. O Apache Kafka suporta diversas linguagens como Java, Python, JavaScript, Ruby, entre muitas outras. Esta ferramenta tem a vantagem de cada tópico ter propriedades dos dois modelos de funcionamento anteriormente falados, permite escalar o processamento pelos consumidores, por

exemplo através do balanceamento da carga por esses consumidores, mas também permite a partilha da mesma mensagem por vários consumidores[49].

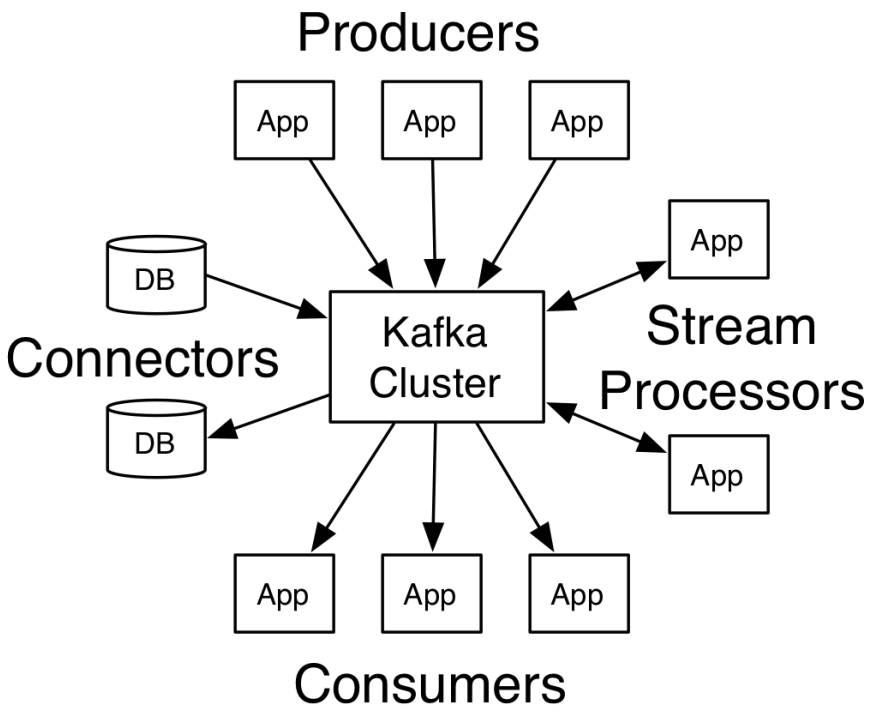

Figura 12 - Arquitetura de Apache Kafka[50]

3.3.2. Open Message Queue

A plataforma *open-source* Open Message Queue (MQ) é "um *middleware* completamente orientado a mensagens"[51]. Isto é, uma plataforma intermediária que fornece serviços orientados a transmissão de mensagens. Implementa um serviço confiável de transmissão de mensagens que permite que as aplicações comuniquem entre si sem depender de comunicações síncronas, providenciando para tal *buffering* entre os produtores de

[49] https://kafka.apache.org/intro.html
[50] Figura retirada de https://kafka.apache.org/23/images/kafka-apis.png
[51] https://javaee.github.io/openmq/

mensagens e os consumidores. OpenMQ implementa a API do *Java Message Service*, uma API em Java que permite criar, enviar, receber e ler mensagens através de um conjunto de interfaces e semânticas associadas[52] e fornece ainda opções como transmissão de mensagens através de SOAP/HTTP[53], distribuição escalável de mensagens, suporte extensivo de JMX (*Java Management Extensions)*, entre outros[54]. O modelo de funcionamento é representado na Figura 13.

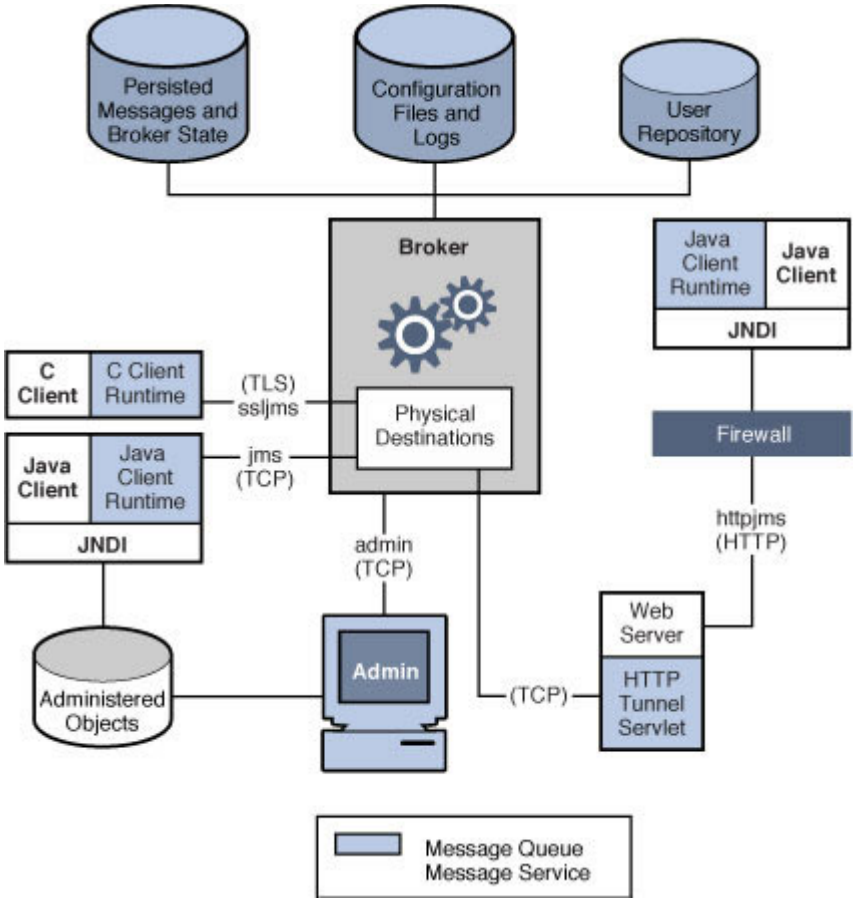

Figura 13 - Arquitetura do OpenMQ[55]

[52] https://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html
[53] Utilização do protocolo de mensagens SOAP, formatadas em XML, transmitidas sobre o protocolo HTTP
[54] https://javaee.github.io/openmq/Overview.html
[55] Figura retirada de https://javaee.github.io/openmq/Overview.html

3.3.3. RabbitMQ

O RabbitMQ é uma das mais populares plataformas *open-source* de transmissão de mensagens. É leve e fácil de implementar quer na *Cloud* quer numa máquina local (*on-*premisses). Pode ser utilizado em configurações distribuídas (pertencem à mesma entidade) e federadas (referentes a diferentes entidades organizacionais) para obter requisitos de alta escalabilidade e alta disponibilidade[56]. Suporta mensagens assíncronas, pode ser executado através de Docker, estando disponíveis imagens para tal, ou configurado usando Puppet, entre outros mecanismos. Foi desenvolvido para suportar a troca de mensagens entre aplicações escritas em várias linguagens como PHP, Python, JavaScript e muitas outras. Permite a sua execução distribuída, através de *clusters* de alta disponibilidade e alta taxa de transferência e fornece ainda muitos extras (*plugins*) e ferramentas, bem como mecanismos de gestão e monitorização. Para empresas e aplicações em *Cloud* permite ainda autenticação, autorização e suporta TLS e LDAP[57].

O RabbitMQ funciona sobre o protocolo AMPQ (Advanced Message Queuing Protocol)[58], um protocolo de mensagens da camada de aplicação do modelo OSI que permite a comunicação entre as aplicações e os *middlewares* com função de *message broker*. Para transmissão de mensagens é utilizado o protocolo TCP, para garantir a certeza de entrega, podendo ser utilizado TLS para proteção do conteúdo das mensagens. No RabbitMQ, como ilustrado na Figura 14, os serviços que produzem mensagens publicam as mensagens para um mecanismo denominado *exchange*, presente no servidor de RabbitMQ. Este mecanismo pode ser visto como caixas de correios e têm a função de receber as mensagens dos *publishers*. De seguida distribuem réplicas das mensagens recebidas pelas várias filas (*queues*) através de regras denominadas *bindings*. Por usa vez, depois das mensagens estarem nas *queues*, os serviços consumidores poderão ter acesso a elas. O protocolo AMQP tem a noção do mecanismo *message acknowledgement*, um mecanismo que

[56] https://www.rabbitmq.com/
[57] https://www.rabbitmq.com/#features
[58] https://www.amqp.org/

permite ao consumidor comunicar ao servidor de RabbitMQ que recebeu a mensagem. Este mecanismo pode ser ativado automaticamente, ou posteriormente, quando o serviço consumidor entender, através de código. O RabbitMQ fornece ainda um *plugin* de gestão e manutenção para Web[59].

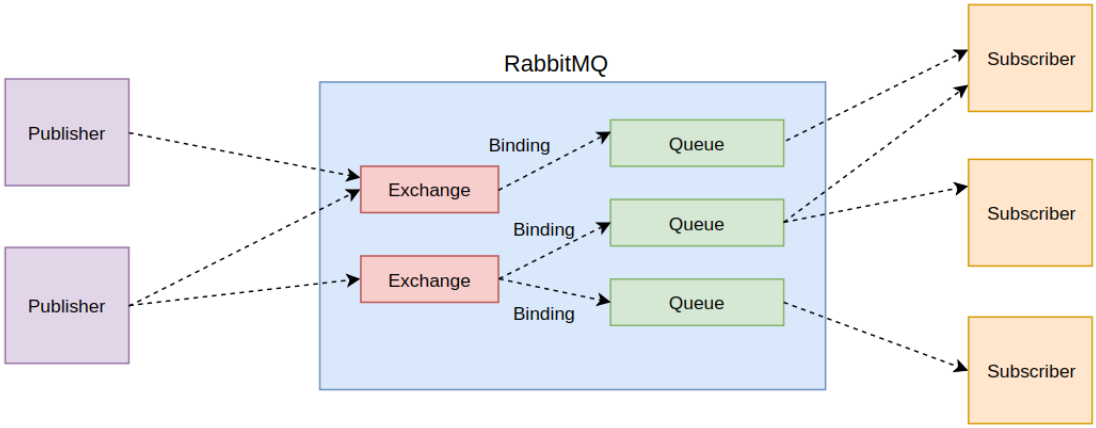

Figura 14 - Arquitetura geral do RabbitMQ[60]

3.3.4. Escolha da Tecnologia

A escolha do mecanismo de comunicação a utilizar recaiu sobre a plataforma RabbitMQ. Diversas razões foram consideradas nesta escolha, desde logo, por ser uma das plataformas mais conhecidas e utilizadas hoje, mas também por ser robusta, num estado de maturidade já algo avançado, bem documentada e fácil de implementar e assente num protocolo robusto, o AMQP. Ambas as plataformas Apache Kafka e RabbitMQ seriam boas escolhas a ter em conta, uma vez que oferecem performance e escalabilidade, porém, devido a já conhecer um pouco da tecnologia RabbitMQ, a escolha recaiu sobre essa. Apache Kafka

[59] https://www.rabbitmq.com/tutorials/amqp-concepts.html
[60] Figura retirada de https://stackabuse.com/spring-cloud-stream-with-rabbitmq-message-driven-microservices/

seria uma boa solução para o projeto se fosse necessário que as mensagens tivessem de estar guardadas durante um largo espaço de tempo (Dobbelaere & Sheykh Esmaili).

3.4. Linguagens Utilizadas nos Microsserviços

Depois de ser escolhida a arquitetura de microsserviços, torna-se necessário escolher as linguagens e tecnologias para começar a desenvolver esses serviços. Existem várias linguagens que poderiam ter sido utilizadas, porém, as várias linguagens que existem têm muitas vezes âmbitos diferentes, isto é, existem linguagens cujo foco é mais académico, como o MATLAB, linguagens interpretáveis mais utilizadas no desenvolvimento Web como o Ruby, outras como diversas bibliotecas que facilitam o seu uso em aprendizagem computacional ou análise de dados, como o Python, e outras mais viradas para aplicações nativas e de grande desempenho, como é o exemplo de C++, Java ou .NET. Uma linguagem pode ser muito boa para desenvolver uma aplicação em servidor, mas não ser tão boa para fazer análise e tratamento de dados. Como referido anteriormente, para este projeto são necessários três serviços com os objetivos distintos: extrair o vídeo da plataforma YouTube e converter para áudio, extração de características do sinal áudio e classificação. Para o desenvolvimento destes serviços foram consideradas várias linguagens de acordo com os seguintes parâmetros: facilidade na manipulação e tratamento de dados e gráficos, performance, facilidade de implementação, documentação e suporte da comunidade. Além disso, é necessário ter em conta os paradigmas de programação cada linguagem.

3.4.1. Python

Python é uma linguagem interpretada (quer dizer que não é compilada, logo é mais lenta) *open-source* e multiparadigmática, gerida pela organização Python Software Foundation, muito utilizada em programação em áreas científicas. A sua primeira aparição foi por volta do ano 1990, criada por Guido van Rossum e teve o seu primeiro lançamento em 1991. Uma das grandes vantagens de Python é a sua aceitação e utilização no meio académico, o

que ao longo do tempo tem feito com que várias bibliotecas para a área de processamento de dados, manipulação de informação, *data science*, *machine learning*, visualização de informação e muitas mais tenham sido desenvolvidas para esta linguagem. A característica mais diferenciadora em relação a outras linguagens é a sua sintaxe limpa e bem organizada e o facto de ter um pacote que é extremamente utilizado em computação científica, o NumPy. Esta ferramenta oferece várias funcionalidades como manipulação de *arrays* com N dimensões, integração com código Fortran, C/C++, capacidades de álgebra linear, transformadas de Fourier entre muitas outras[61]. A junção das características e das imensas bibliotecas existentes para Python, permitem que esta seja uma boa solução para *machine learning.*

3.4.2. JavaScript

JavaScript é uma linguagem de programação interpretável multiparadigmática criada por Brendan Eich em 1995 e desenvolvida pela Netscape Communications Corporation, a Mozilla Foundation e pela Ecma International. Tornou-se popular como linguagem de *scripting* para correr no *browser*, porém o JavaScript é também utilizado em "muitos ambientes além dos navegadores, tais como Node.JS, Apache CouchDB e Adobe Acrobat"[62], e é uma das linguagens mais importantes que atualmente existem. Nos últimos anos surgiram iniciativas para executar do lado do cliente (Node.JS) e surgiu o JIT [*Just in Time*] *compiler* que fez explodir também como solução para desenvolvimento de *Backend* e não só de *Frontend*. O JavaScript não é uma linguagem desenvolvida para ser utilizada para fins científicos[63], porém oferece algumas ferramentas que podem ser utilizadas para este fim. Como o Python, o JavaScript oferece muitos pacotes e bibliotecas que podem ser instaladas e geridas através do gestor de pacotes de JavaScript denominado npm (Node Package Manager). Em termos de performance, o JavaScript é muito rápido (comparado

[61] https://numpy.org/
[62] https://developer.mozilla.org/pt-PT/docs/Web/JavaScript
[63] https://link.springer.com/article/10.1186/s13321-019-0331-1#Sec6

com outras linguagens interpretadas) a executar aplicações pouco complexas, mas com o aumento da complexidade de cálculos, a linguagem vai perdendo performance[64]. Outra grande mais-valia do JavaScript é o facto de ser a linguagem com a maior distribuição, superando o Java–[65], tendo, inclusive, dado origem à linguagem ECMAScript[66], uma linguagem criada pela Ecma International para padronizar o JavaScript. Esta linguagem tem uma das maiores comunidades e documentações devido a ser uma tecnologia Web[67].

3.4.3. Ruby

"Ruby é uma linguagem interpretada dinâmica, *open-source* com foco na simplicidade e na produtividade. Tem uma sintaxe elegante, de leitura natural e fácil escrita"[68]. Criado por Yukihiro "Matz" Matsumoto, o Ruby surgiu em 1995 com objetivo do seu criador juntar algumas partes das suas linguagens favoritas, Perl, Smalltalk, Eiffel, Ada e Lisp. Matz tentou criar uma linguagem tao natural quanto simples e acrescentou que "o Ruby é simples na aparência, mas muito complexo no interior, tal como o corpo humano"[69]. A grande expansão desta linguagem deve-se em muito à popularidade da *framework* de arquitetura MVC (Model-View-Controller) Ruby on Rails. Ruby assenta no princípio de que tudo é um objeto e é uma linguagem extremamente flexível, isto é, para o programador, é possível adicionar, alterar ou remover partes da linguagem à medida que seja necessário. Ruby teve uma grande adoção junto das grandes *startups* tecnológicas que deram origem ao Twitter, GitHub e várias outras, pois a linguagem com a *framework* Ruby on Rails permitia, num curto espaço de tempo, desenvolver uma aplicação web robusta e funcional, seguindo MVC. Atualmente, a linguagem está mais madura e funcional, mas perdeu folego, fruto do seu baixo desempenho quando comparado com outras opções que

[64] https://link.springer.com/article/10.1186/s13321-019-0331-1#Sec7
[65] https://link.springer.com/article/10.1186/s13321-019-0331-1#sec8
[66] https://www.ecma-international.org/memento/tc39.htm
[67] https://link.springer.com/article/10.1186/s13321-019-0331-1#Sec13
[68] http://www.ruby-lang.org/pt/
[69] http://www.ruby-lang.org/pt/about/

apareceram agora, como Node.JS com a *framework* Express[70] e o paradigma SPAs com APIs.

3.4.4. MATLAB

MATLAB é um *software* pago e de código fechado que "combina um ambiente de Desktop ajustado para análise iterativa e processos de design com uma linguagem de programação que expressa diretamente matrizes e *arrays* matemáticos"[71]. Este *software* "integra computação, visualização e programação num ambiente fácil de utilizar onde os problemas e soluções são expressas em notação matemática familiar. Tipicamente, é utilizada em matemática e computação, desenvolvimento de algoritmos, modelação, simulação e prototipagem, análise de dados, exploração e visualização, gráficos científicos ou de engenharia, desenvolvimento de aplicações, incluindo construção através de *Graphical User Interface* (GUI)"[72].

3.4.5. Considerações finais

As linguagens referidas anteriormente são ótimas escolhas para serem utilizadas no projeto, porém cada uma apresenta as suas vantagens e desvantagens. No que toca às tarefas de processamento de sinal e classificação, uma escolha possível para fazer um trabalho de investigação (local, não distribuído) seria por exemplo o MATLAB, devido à panóplia de funcionalidades que oferece para a comunidade científica e estudantil. Porém, utilizar MATLAB num microsserviço seria extremamente difícil e também muito dispendioso em termos monetários e de peso computacional. Entre as outras três opções, optou-se pela utilização de Python para os microsserviços que tinham como objetivo a

[70] https://expressjs.com/
[71] https://www.mathworks.com/products/matlab.html
[72] https://cimss.ssec.wisc.edu/wxwise/class/aos340/spr00/whatismatlab.htm

extração de características musicais e também a classificação do sinal áudio devido à facilidade de tratamento de dados que o Python oferece. Embora a linguagem seja interpretada, utiliza diversas bibliotecas que na sua base são otimizadas com código C e C++ aumentando a performance. Já para o microsserviço de extração do vídeo/música e conversão para áudio foi utilizada a linguagem JavaScript devido à grande quantidade de pacotes e módulos que existem para extração de vídeos do YouTube.

Capítulo 4

Desenvolvimento da Solução

A solução planeada para o problema proposto aborda diversos tópicos complexos e até desconhecidos. Diversos fatores contribuem para isto. Primeiro, o facto de o reconhecimento emocional em música ser um assunto ainda em investigação, que une aprendizagem automatizada e processamento de sinal. Por outro lado, tópicos como desenvolvimento de microsserviços, filas de mensagens ou orquestradores de *containers*, são também eles tópicos bastante atuais.

Por estas razões o projeto seguiu uma metodologia de desenvolvimento ágil, contando com uma fase inicial de forte experimentação e prototipagem. Esta fase permitiu testar as diversas abordagens a seguir e aferir a viabilidade das mesmas, mitigando o possível risco de uma falha numa fase mais avançada do desenvolvimento.

Numa fase inicial foram testadas tecnologias de gestão de *containers*, sendo o Docker a mais explorada.

Fez-se uma pequena aplicação para testar o funcionamento do Docker, na qual foi possível criar um enxame ligando várias máquinas virtuais num cluster. Estas máquinas virtuais, onde o Docker foi executado, foram criadas através do Virtualbox, outro sistema de virtualização. O Docker permite a criação de *hosts* (máquinas virtuais) que executam Docker através da ferramenta *docker-machine*, o que poderia ter sido utilizado em alternativa ao Virtualbox. Porém, o Docker para Windows tem de ser utilizado com o Hyper-V (tecnologia de virtualização da Microsoft) ativo e esta funcionalidade entra em conflito com o Virtualbox, outro *software* de virtualização. Uma vez que o Virtualbox foi utilizado para prototipagem dos microsserviços, havendo já experiência prévia com este (como explicado de seguida), optámos por substituir o *docker-machine* por este.

Após perceber o funcionamento do Docker, optámos por utilizar nos protótipos seguintes uma tecnologia com a qual já havia experiência – Vagrant[73]. Esta serve para produzir de forma automatizada diversos ambientes aplicacionais de desenvolvimento (através da criação de máquinas virtuais), abstraídos uns dos outros, facilmente reproduzidos e com grande portabilidade. O Vagrant foi utilizado neste projeto para desenvolver e alojar os protótipos dos microsserviços, simulando os *containers* no Docker, durante a fase de desenvolvimentos.

Posteriormente foram criadas várias máquinas virtuais, utilizando o Vagrant, que serviram para implementar cada uma das tarefas do sistema de forma isolada. Primeiro sob a forma de simples provas de conceito e depois evoluindo para os microsserviços responsáveis pelas diversas tarefas:

- Extração do vídeo a partir do YouTube
- Conversão do vídeo para áudio
- Extração de características musicais
- Classificação emocional
- Sistema de intermediação de mensagens

Os detalhes sobre estes serviços, cuja interligação é ilustrada na Figura 15, são apresentados nas secções seguintes. A última fase, passa pela instalação do Docker num servidor e fazer implementação dos microsserviços e do servidor de gestão de mensagens em *containers* no Docker.

[73] https://www.vagrantup.com/

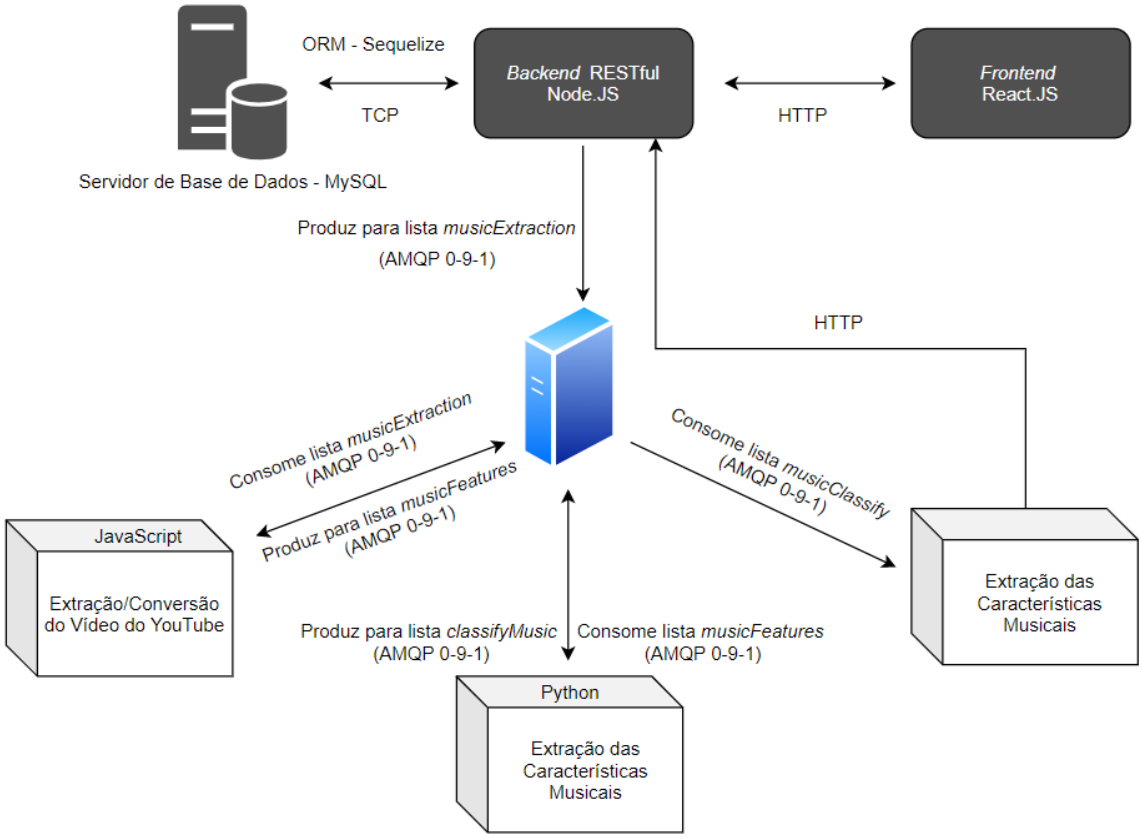

Figura 15 - Solução Final com tecnologias utilizadas

4.1. Aplicação Web

O contacto dos utilizadores com o sistema de reconhecimento emocional acontece através de uma interface Web, representada na Figura 16. Neste, o utilizador pode consultar as músicas que já foram processadas antes e que existem na base de dados. Também permite introduzir uma nova música para ser classificada (através do *link* para o YouTube) e gerir a sua lista de músicas classificadas. Para além disto, é possível gerir a informação do sistema e ver o estado dos serviços através de um painel de administração.

Figura 16 - Página principal da aplicação Web

Esta aplicação web está fora do âmbito deste trabalho de mestrado, tendo sido desenvolvida pelos alunos de projeto final de licenciatura Tiago António e Tiago Areias e por isso não é abordado em detalhe neste relatório. Como referido na Figura 17, consiste num sistema de informação desenvolvido com Node.JS para o *backend* e React.JS para o *frontend*, comunicando com o sistema de reconhecimento de emoção através do intermediário de mensagens.

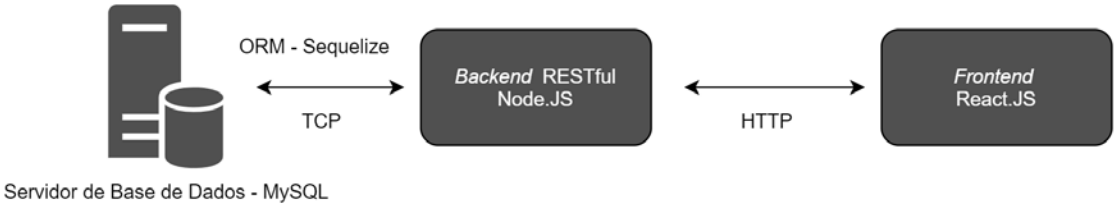

Figura 17 - Arquitetura da aplicação Web

4.2. Comunicação entre Microserviços

O serviço de gestão de mensagens, em RabbitMQ, é responsável pela gestão das comunicações entre os microsserviços e por receber os pedidos de processamento vindos da aplicação Web quando um utilizador insere um novo vídeo ou música. Os microsserviços consumidores de mensagens, como é o caso do de extração do vídeo e conversão para áudio, de extração de características musicais do sinal áudio e o de classificação emocional estão sempre conectados com o servidor de mensagens, através de canais específicos. Estes vão consumindo mensagens assim que estas são publicadas numa das filas em que são responsáveis. Atualmente são usados três canais de mensagens:

- Fila *musicExtraction*
- Fila *musicFeatures*
- Fila *classifyMusic*

O microsserviço de extração do vídeo e sua conversão para áudio é responsável por consumir a fila "*musicExtraction*", o de extração de características é responsável por consumir fila "*musicFeatures*" e o de classificação é responsável por consumir a fila "*classifyMusic*". Todos estes microsserviços, à exceção do de classificação, têm também a função de publicar o resultado do trabalho realizado para as filas do servidor. Isto é, o microsserviço de extração de música tem como objetivo retornar o ID do áudio que foi extraído e o microsserviço de extração de características tem como objetivo retornar o ID

do áudio e um conjunto de características e seus valores. Já o microsserviço de classificação faz o registo da classificação diretamente na base de dados da aplicação Web. Este processo é ilustrado na figura abaixo.

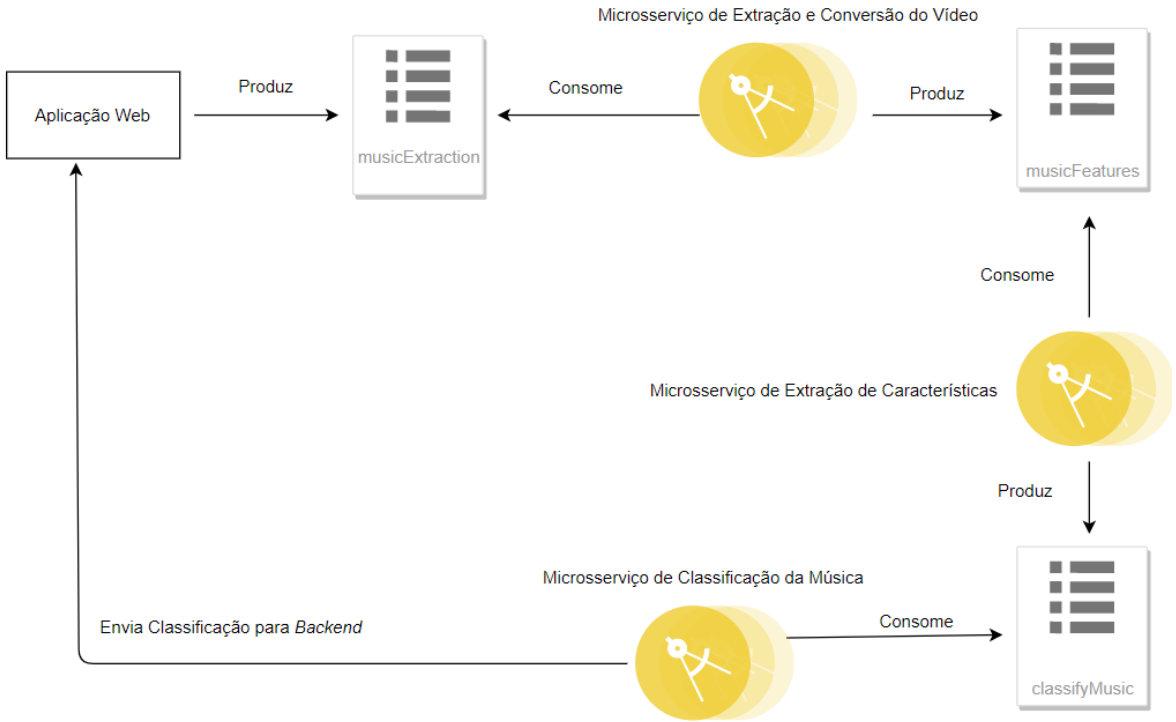

Figura 18 - Fluxo de produção e consumo dos microsserviços

4.3. Extração de Vídeo e Conversão Áudio

O microsserviço para a extração do vídeo de uma página do YouTube, separação do canal de áudio e conversão do mesmo foi desenvolvido em JavaScript com recurso à biblioteca ytdl-core[74]. Esta biblioteca em JavaScript disponibiliza um conjunto de funções para interagir com o serviço de vídeos do YouTube. Entre estas, permite a extração do vídeo e sua conversão para diferentes formatos, mas também outras funcionalidades com validação

[74] https://github.com/fent/node-ytdl-core#readme

do URL do vídeo, extração da informação (meta-dados) do vídeo, filtrar com base em formatos específicos, entre outras funcionalidades.

A comunicação com o servidor de mensagens neste microsserviço foi desenvolvida com recurso à biblioteca "ampqlib"[75]. A biblioteca ampqlib é sugerida na página oficial do RabbitMQ e implementa os mecanismos necessários para comunicação através do protocolo AMQP 0-9-1.

A sequência de operações realizadas pelo serviço de extração de vídeos é descrita de seguida. O serviço aguarda por novas mensagens na fila "*musicExtraction*" do servidor RabbitMQ. Ao receber uma mensagem, consome a mesma e começa por validar o URL que vem na mesma para garantir que indica um vídeo do YouTube. Caso o mesmo seja válido, o vídeo é então descarregado e depois convertido para o formato WAV (Waveform Audio File Format). A principal razão para a conversão para WAV é a de manter a coerência com o procedimento descrito na literatura. Por norma os ficheiros originais são convertidos para formatos mais simples, por exemplo de MP3 stereo com 44.1kHz de amostragem para WAV mono a 22kHz, por forma a aligeirar o processo de processamento de sinal. Depois de obter o áudio, este é gravado num ficheiro com o nome "<*idVideo*>.*wav*". É então enviada uma mensagem com este ID para a fila "*musicFeatures*", sendo retornada uma confirmação da receção da mesma. Após este processo o serviço volta ao estado de espera por novas mensagens. O retorno da confirmação da receção é uma forma de garantir que a mensagem foi processada da forma correta e que correu tudo como esperado. Este processo é ilustrado na Figura 19.

[75] https://www.squaremobius.net/amqp.node/

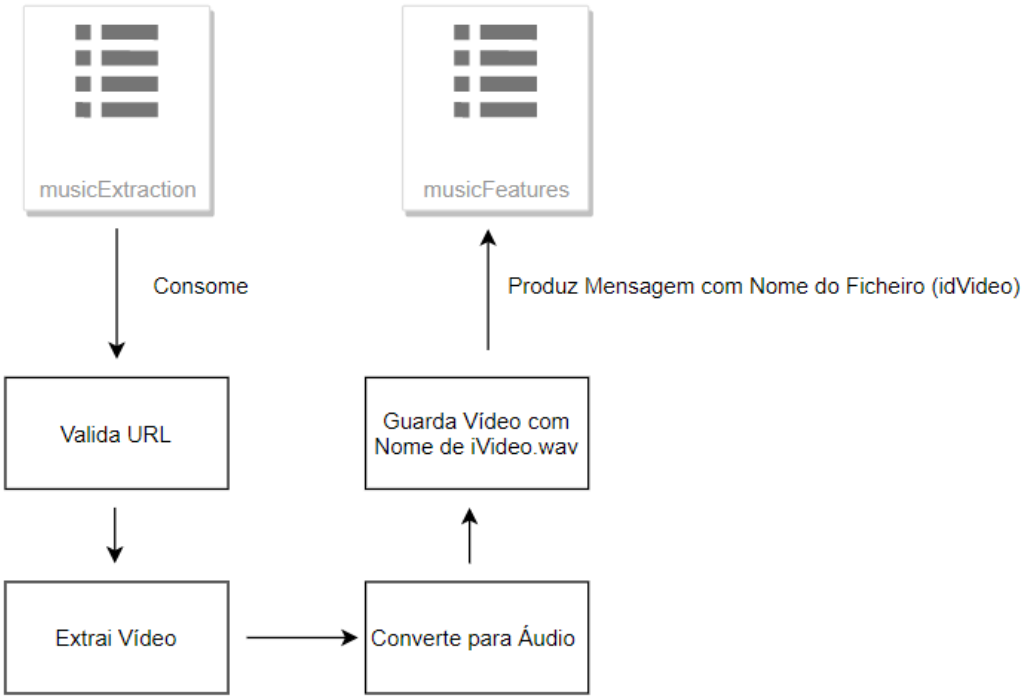

Figura 19 - Diagrama de funcionamento do microsserviço de extração e conversão de vídeo

4.4. Extração de Características Musicais

O microsserviço de extração de características dos sinais áudio foi desenvolvido em Python e foram implementadas abordagens com duas bibliotecas de processamento de sinal para obter uma comparação acerca da simplicidade e eficácia de cada uma, a LibROSA e a Essentia (analisadas na secção 2.4.2). A biblioteca Essentia oferece uma maior quantidade de funcionalidades que tornaram muito simples a extração de características. É também uma biblioteca bastante eficiente em termos computacionais, como abordado anteriormente, e por esta razão foi adotada como a biblioteca principal. Foi feito também um pequeno teste para recolher os batimentos por minuto através da LibROSA.

Este microsserviço tem um canal criado com o servidor de RabbitMQ para consumir a lista "*musicFeatures*". Cada vez que aparece uma nova mensagem, composta pelo ID da música

para a qual se pretende extrair as características, o microsserviço utiliza a biblioteca Essentia para a extração de:

- *average_loudness* – o volume médio da música,
- *pitch_salience_mean* – a média da "força" das notas, por exemplo, agudas ou graves (quão salientes são as notas em relação ao restante som).
- *rhythm.bp*m – informação sobre o número de batimentos por minuto

A *framework* é capaz de extrair um número bastante maior de características do sinal áudio, o que deverá ser feito na versão de produção do sistema. No entanto, dado o tempo necessário para extrair dezenas ou centenas destas, durante o desenvolvimento optámos por utilizar apenas algumas. Depois de extraídas as características, o microsserviço faz a publicação para a lista "*classifyMusic*" de uma mensagem contendo uma tabela com o ID do áudio, e as três características referidas anteriormente. Para uma fase posterior, tenciona-se fazer a recolha de todas as características que a biblioteca Essentia proporciona, tornando ainda melhor os resultados da classificação. No teste da biblioteca LibROSA, foi utilizada a características *beat_times* que corresponde aos batimentos da música. Este processo de funcionamento do microsserviço de extração de características é ilustrado na Figura 20.

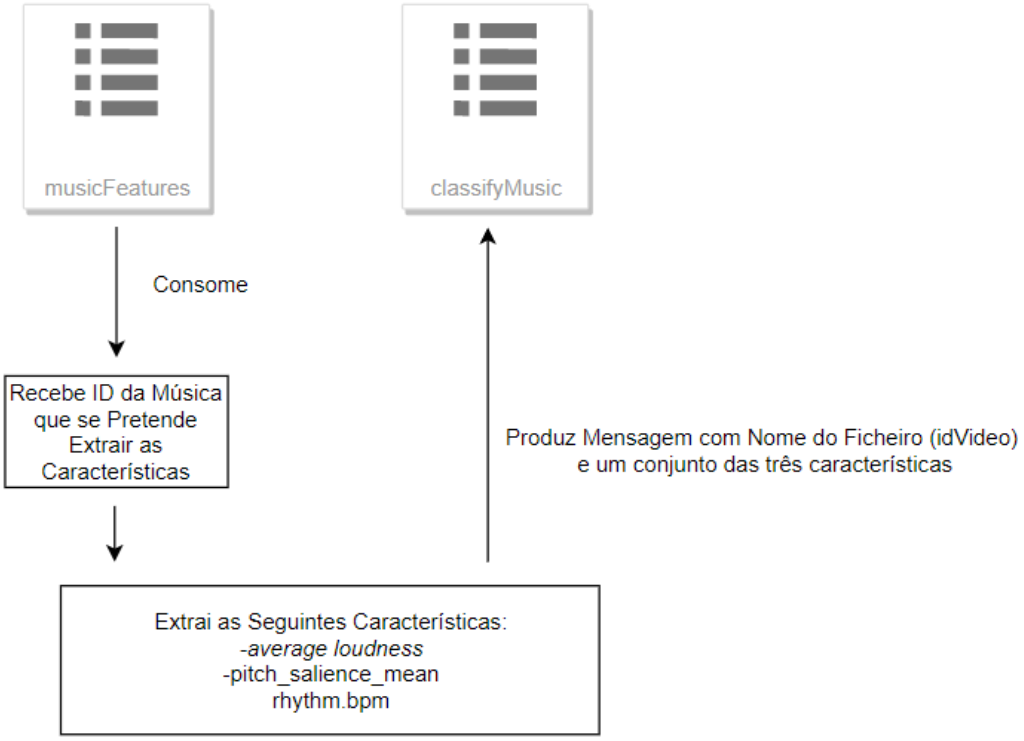

Figura 20 - Diagrama de funcionamento do microsserviço de extração de características

Para a extração das características do *dataset* (Panda, Malheiro, & Paiva, 2018) (conjunto de dados de 900 músicas e respetivas anotações categóricas) utilizado na fase de treino do modelo de classificação, foi criado um pequeno *script* que faz a extração das três características anteriores para todos os ficheiros e guarda as mesmas num ficheiro com extensão CSV (Comma Separated Values). O ficheiro resultante é composto por *N* linhas que representam as *N* músicas, em que cada linha contém os valores das três características para a música em causa, assim como a classe a que pertencem. Com esta informação, é possível retreinar o modelo sem que seja necessário voltar a extrair as características anteriormente mencionadas. O processo é ilustrado na Figura 21.

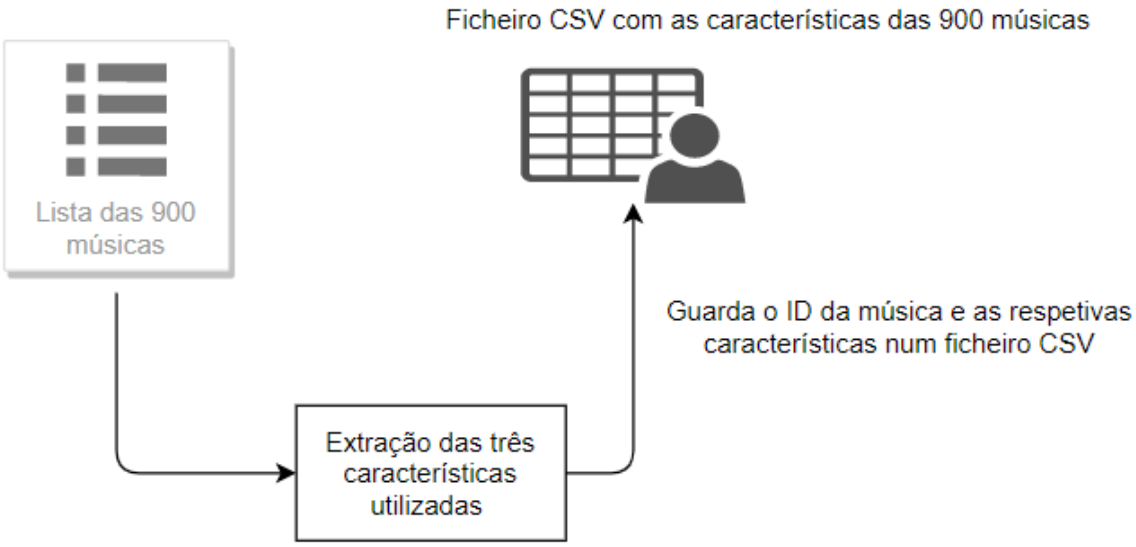

Figura 21 - Processo de extração das características das músicas do *dataset*

A biblioteca utilizada para fazer a comunicação com o servidor de fila de mensagens é denominada de pika[76], uma biblioteca em Python para suporte do protocolo AMQP 0-9-1 e sugerida pela documentação oficial do RabbitMQ.

4.5. Classificação Emocional do Sinal Áudio

O microsserviço de classificação de emoção foi desenvolvido em Python e utiliza a implementação do classificador SVM (analisado na secção 2.4.3) disponível na biblioteca Scikit-Learn, referenciada anteriormente na secção 0.

4.5.1. Treino do Modelo de Classificação

Antes do desenvolvimento deste microsserviço, que tem como função a de prever a emoção de sinal desconhecido, foi necessário implementar um pequeno *script* para treinar e testar o modelo de classificação a utilizar. Esse *script*, utilizado na fase de treino da

[76] https://pika.readthedocs.io/en/stable/

máquina, percorre o ficheiro CSV que contém as características extraídas pelo *script* de extração de características, descrito acima, utilizado também para a fase de treino. Das 900 músicas existentes no *dataset* são utilizadas 200 pertencentes a cada uma das quatro etiquetas nas quais o *dataset* está repartido (1-Feliz, 2-Tensa, 3-Triste, 4-Calma) e 25 para fase de testes. O critério de escolha do número de elementos utilizados para teste e treino, neste caso, foram de cerca de 10% para teste e 90% para treino. Depois, foi feito um pré-processamento dos dados (características) para que não haja erros devido a escalas dispares entre as diferentes características. O processo do pré-processamento é responsável por normalizar os dados, isto é, transformar os valores de todas as características para um mesmo intervalo, por exemplo [0, 1]. Depois do pré-processamento, é feita uma avaliação exaustiva de parâmetros de configuração do classificador, variando os parâmetros de configuração, neste caso os valores de custo e gama, para tentar obter o melhor modelo para a classificação. Este processo é feito com recurso a uma função da biblioteca Scikit-Learn denominada GridSearchCV. Depois disso, o modelo é treinado com as primeiras 200 músicas de cada uma das 4 classes e guardado num ficheiro para ser utilizado pelo microsserviço de classificação. Além desse modelo, é também guardado o *scaler* resultante do treino dessas características para que seja possível normalizar as características de músicas futuras que o microsserviço irá analisar, dentro das escalas que foram usadas durante o treino. Este processo é ilustrado na Figura 22.

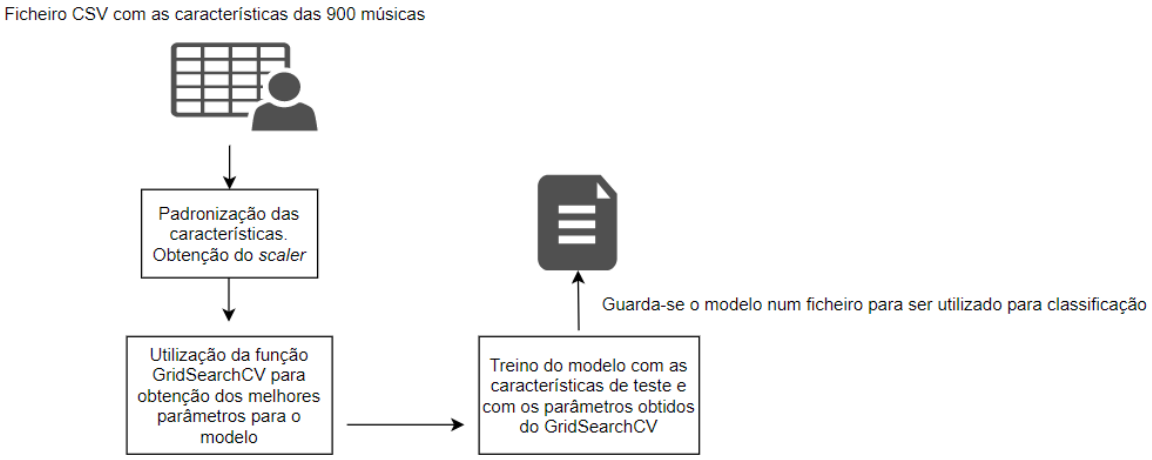

Figura 22 - Processo de treino do modelo e obtenção do *scaler*

4.5.2. Processo de Previsão de Emoção

O programa principal de classificação é responsável por consumir as mensagens que surgem na lista "*classifyMusic*", através da biblioteca pika, anteriormente referida. Cada mensagem é composta pelo ID da música e as características do sinal áudio que a representa e para as quais se pretende obter uma classificação. Essas características são padronizadas através do *scaler* gerado anteriormente e depois, com auxílio da função *predict*, o modelo obtém-se uma classificação para as novas características com base nos padrões previamente encontrados no *dataset* de treino. Esta classificação representa uma classe de 1 a 4, que são as etiquetas previamente treinadas. O microsserviço finaliza o seu trabalho, antes de voltar ao estado de espera por mensagem, inserindo na base de dados da aplicação o ID do vídeo em causa e a classificação obtida. Este processo é ilustrado na Figura 23.

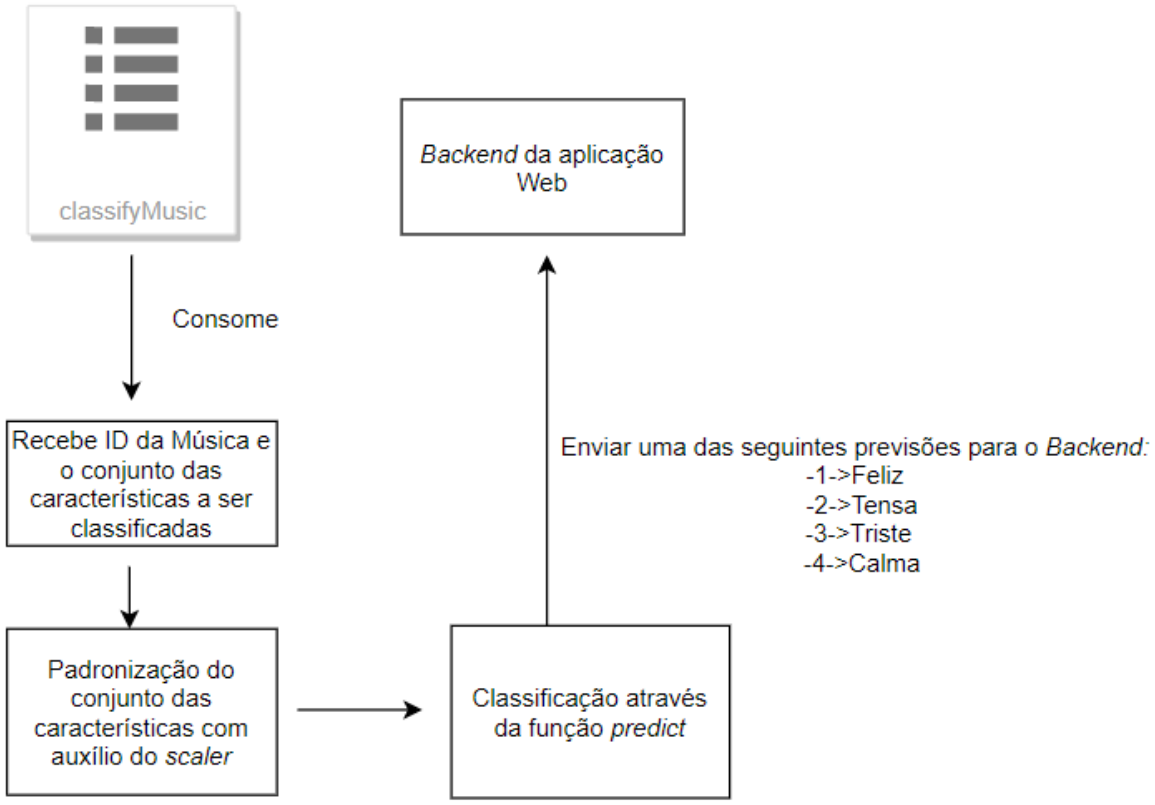

Figura 23 - Diagrama de funcionamento do microsserviço de classificação

4.5.3. Precisão do Sistema de Classificação

Este trabalho teve como foco principal o desenvolvimento de um protótipo de reconhecimento emocional robusto, que mais tarde possa ser expandido com novas abordagens desenvolvidas pelos investigadores na área. Não sendo a precisão de classificação objetivo central, foi ainda assim testada a capacidade do sistema. Deve ser tido em conta que ao contrário do artigo que serviu de inspiração (Panda, Malheiro, & Paiva, 2018), o número de características musicais e o processo de treino são relativamente diferentes e não otimizados.

Para este fim foram extraídas das 900 músicas do *dataset*, 3 características disponíveis no Essentia (características *average_loudness*, *pitch_salience_mean* e *rhythm_bpm*). Estas foram normalizadas e testadas tentando replicar o procedimento do artigo – 20 repetições de validação cruzada com 10 grupos (repeated stratified K fold cross validation com K = 10). Foi obtido um valor médio de precisão de 0.427, com um desvio padrão de 0.047. Este valor é relativamente baixo quando comparado com o artigo original, sendo já expectável dado o baixo número de características utilizadas. Sendo apenas uma prova de conceito, acreditamos que poderá ser melhorado com a inclusão de novas características musicais relevantes e otimizações do serviço de classificação.

4.6. Trabalho Futuro

Sendo uma primeira versão do sistema, existem ainda diversas ideias que podem e devem ser incluídas numa oportunidade futura. Entre estas, destacamos as seguintes:

- Implementação (*deploy*) de todo o projeto através de Docker para que seja robusto e mais facilmente escalável

- Adição de novas características musicais que sejam mais relevantes para o problema de MER
- Utilização de uma base de dados não relacional para guardar as características extraídas e desta forma evitar a extração as mesmas de novo aquando da reclassificação de uma música
- Separação de músicas completas em segmentos menores e classificação dos mesmos, uma vez que na literatura são por norma usados segmentos de 15 a 30 segundos
- Estudar a utilização de vários *containers* de um mesmo microsserviço em modo de enxame ou replicação e mecanismos de escala automática para picos de carga

Capítulo 5
Conclusão

A área do reconhecimento de emoção em música ainda é hoje um problema em aberto, porém, com o atual panorama de empresas e centros de investigação que estão a apostar nesta área é provável que cada vez mais surjam novas abordagens e maneiras inovadoras de reduzir os problemas que hoje existem no processo do reconhecimento. Entre os problemas mais complexos encontramos o facto de não se conseguir chegar a um consenso sobre como definir e identificar uma emoção, o que leva a uma grande dificuldade na criação de conjuntos de dados de larga escala. Por outro lado, é difícil saber quais as melhores características que devemos extrair de um sinal áudio para que possamos ter uma boa classificação, sendo que ainda são necessárias características novas que sejam mais relevantes para a área de MER.

O principal objetivo deste projeto passou por conseguir integrar todas as etapas de tratamento de um sinal áudio para identificação de emoções, para que fosse proporcionado aos utilizadores uma nova experiência de utilização. O desenvolvimento deste sistema passou pela utilização de tecnologias e paradigmas que são atualmente estado da arte, permitindo ter um sistema escalável e robusto, menos propício a erros e a tempos de espera elevados. A utilização de microsserviços interligados por uma fila de mensagens proporciona uma capacidade de resposta atempada e sem falhas aos pedidos do cliente, não ficando o sistema parado em cada uma das tarefas, que no caso de MER demoram imenso tempo a serem concluídas.

Com isto, colmatou-se o facto de haver muito poucas plataformas robustas a demonstrar este conceito e espera-se também que contribua para o desenvolvimento desta área que é o reconhecimento de emoções em música.

Referências

Alpaydin, E. (2010). *Introducing to Machine Learning.* Retrieved 09 13, 2019, from https://books.google.pt/books?hl=pt-PT&lr=&id=TtrxCwAAQBAJ&oi=fnd&pg=PR7&dq=what+is+machine+learning+paper&ots=T5frNIU5kQ&sig=wzBEwtuVT0-Ke4CY1ScktclLKRY&redir_esc=y#v=onepage&q&f=false

B. Kotsiantis, S. (2007, 07 16). *Supervised Machine Learning: A Review of Classification Techniques*. Retrieved 09 13, 2019, from https://datajobs.com/data-science-repo/Supervised-Learning-[SB-Kotsiantis].pdf

Cardoso, L., Panda, R., & Pedro Paiva, R. (n.d.). *MOODetector: A Prototype Software Tool for*. Retrieved 10 7, 2019, from http://mir.dei.uc.pt/pdf/Conferences/MOODetector/INForum%202011.pdf

Chen, Y.-H. Y. (2012, 05). Machine Recognition of Music Emotion: A Review. p. 30. Retrieved 09 2, 2019, from http://mac.citi.sinica.edu.tw/~yang/pub/yang12tist.pdf

Dobbelaere, P., & Sheykh Esmaili, K. (n.d.). Industry Paper: Kafka versus RabbitMQ. *A comparative study of two industry reference publish/subscribe implementations*. Retrieved 10 11, 2019, from http://www-inf.telecom-sudparis.eu/COURS/CILS-IAAIO/Articles/debs_kafka_versus_rabbitmq.pdf

Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (n.d.). *Microservices: yesterday, today, and tomorrow*. Retrieved 10 8, 2019, from https://arxiv.org/pdf/1606.04036.pdf

E. Kim, Y., M. Schmidt, E., Migneco, R., G. Morton, B., Richardson, P., Scott, J., . . . Turnbull, D. (2010). *MUSIC EMOTION RECOGNITION: A STATE OF THE ART REVIEW*. Retrieved 09 14, 2019, from https://archives.ismir.net/ismir2010/paper/000045.pdf

Ekman, P. (n.d.). *Handbook of Cognition and Emotion.* Retrieved 10 20, 2019, from https://books.google.pt/books?hl=pt-PT&lr=&id=vsLvrhohXhAC&oi=fnd&pg=PA45&dq=Ekman,+P.+(1999).+Basic+emotions.+Handbook+of+cognition+and+emotion,+98(45-60),+16.&ots=uTBJfoVdHh&sig=EN8GANSHs3DEESz1uRxv3HtzunA&redir_esc=y#v=onepage&q&f=false

Expert System. (n.d.). *What is Machine Learning? A definition*. Retrieved 11 5, 2019, from https://expertsystem.com/machine-learning-definition/

Haq, S. u. (2018, 05 2). *Introduction to Monolithic Architecture and MicroServices Architecture.* Retrieved 10 3, 2019, from Medium: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

Hevner, K. (n.d.). *Experimental studies of the elements of expression in music*. Retrieved 10 20, 2019, from https://pdfs.semanticscholar.org/43db/b2af8861ba8294562f4d565553a814de2396.pdf

Juslin, P. N. (2013, 09 6). *What does music express? Basic emotions and beyond*. Retrieved 09 14, 2019, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3764399/

Kalske, M., Makitalo, N., & Mikkonen, T. (2018). *Challenges When Moving from Monolith to*. Retrieved 10 08, 2019, from https://helda.helsinki.fi/bitstream/handle/10138/237054/challenges_moving_monolith.pdf?sequence=1

Malheiro, R., Panda, R., Gomes, P., & Pedro Paiva, R. (n.d.). *Emotionally-Relevant Features for Classification and Regression of Music Lyrics*. Retrieved 10 25, 2019, from https://ieeexplore.ieee.org/abstract/document/7536113

McFee, B., Raffel, C., Liang, D., P.W. Ellis, D., McVicar, M., Battenberg, E., & Nieto, O. (2015). *http://conference.scipy.org/proceedings/scipy2015/pdfs/brian_mcfee.pdf*. Retrieved 09 12, 2019, from http://conference.scipy.org/proceedings/scipy2015/pdfs/brian_mcfee.pdf

Moffat, D., Ronan, D., & D. Reiss, J. (2015). *AN EVALUATION OF AUDIO FEATURE EXTRACTION TOOLBOXES*, pp. 1-2. Retrieved 09 11, 2019, from https://www.ntnu.edu/documents/1001201110/1266017954/DAFx-15_submission_43_v2.pdf

Oliphant, T. E. (2007). *Python for Scientific Computing*. Retrieved 10 2019, from https://pdfs.semanticscholar.org/35ce/eedd93e2b42f5e2412b1f6e88dd6d5d4679d.pdf

Panda, R. E. (2019). *Emotion-based Analysis and Classification of Audio Music.* Tese de Doutoramento, Faculdade de Ciências e Tecnologias da Universidade de Coimbra, Departamento de Engenharia Informática. Retrieved 09 11, 2019, from https://www.cisuc.uc.pt/publication/show/5806

Panda, R., & Pedro Paiva, R. (n.d.). *Automatic creation of mood playlists in the thayer plane: A methodology and a comparative study*. Retrieved 10 2019, from https://www.researchgate.net/profile/Rui_Pedro_Paiva/publication/257308136_Automatic_creation_of_mood_playlists_in_the_thayer_plane_A_methodology_and_a_comparative_study/links/00b49524e01263a121000000.pdf

Panda, R., Malheiro, R., & Paiva, R. P. (2018, 03). *Novel audio features for music emotion recognition*. Retrieved 09 04, 2019, from https://www.cisuc.uc.pt/publication/show/5509

Pedregosa, F., Varoquaux, G., Gramfort , A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (n.d.). *Scikit-learn: Machine Learning in Python*. Retrieved 11 2019, from http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

Villamizar, M., Castro, H., Verano Merino, M., & Casallas, R. (2015, October). *Evaluating the monolithic and the microservice architecture pattern to deploy*. Retrieved 10 3, 2019, from https://www.researchgate.net/profile/Mario_Villamizar/publication/304317852_Evaluating_the_monolithic_and_the_microservice_architecture_pattern_to_deploy_web_applications_in_the_cloud/links/5b3ad04ca6fdcc8506ea541b/Evaluating-the-monolithic-and-the-micros

Yang, Y.-H., Lin, Y.-C., Su, Y.-F., & H. Chen, H. (n.d.). A Regression Approach to Music Emotion Recognition. Retrieved 11 2019, from https://ieeexplore.ieee.org/abstract/document/4432654/authors#authors