

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Paulo Kiyoshi Oyama Filho

**Sistema de Recomendação Híbrido para  
Músicas Baseado em Metadados**

**Uberlândia, Brasil**

**2024**

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Paulo Kiyoshi Oyama Filho

**Sistema de Recomendação Híbrido para Músicas  
Baseado em Metadados**

Trabalho de conclusão de curso apresentado  
à Faculdade de Computação da Universidade  
Federal de Uberlândia, como parte dos requi-  
sitos exigidos para a obtenção do título de  
Bacharel em Ciência da Computação.

Orientador: Pedro Franklin

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2024

Trabalho de conclusão de curso apresentado  
à Faculdade de Computação da Universidade  
Federal de Uberlândia, como parte dos requi-  
sitos exigidos para a obtenção do título de  
Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 13 de Abril de 2024:

---

**Pedro Franklin**  
Orientador

---

Renan G. Cattelan  
**Professor**

---

Rodrigo S. Miani  
**Professor**

Uberlândia, Brasil  
2024

# Agradecimentos

*Agradeço, primeiramente a Deus,*

*E, não me esquecendo também de toda a minha família e amigos que me ajudaram a chegar até aqui, e puderam tornar realidade a conclusão de mais esta etapa.*

*‘Mas agora, assim diz o SENHOR que te criou, ó Jacó, e que te formou, ó Israel: Não temas, porque eu te remi; chamei-te pelo teu nome, tu és meu.’ Isaías 43:1*

# Resumo

A quantidade de conteúdo gerado a cada dia tem crescido rapidamente, impulsionada pelo contínuo avanço tecnológico e pela sua natureza dinâmica. A partir deste cenário, surge um desafio: desenvolver sistemas de recomendação que consigam acompanhar o constante fluxo de dados criados a todo instante. Os algoritmos de recomendação devem ser assertivos em suas escolhas com base nos dados disponíveis e, ao mesmo tempo, devem ter a capacidade de ajustarem-se à introdução de novos conteúdos. Entretanto, esses tipos de sistemas são escassos na literatura, às vezes limitando-se apenas à concepção teórica sem a correspondente implementação prática, ou vice-versa.

Dessa forma, esta monografia tem como objetivo a construção de um sistema de recomendação para músicas, detalhando a teoria matemática, estatística e computacional adotada em cada etapa do projeto. Para isso, a API (*Application Programming Interface*) pública do *Spotify* será acessada a partir de técnicas de *scrapping* para a coleta dos dados. Os dados obtidos serão analisados, tratados, manipulados e transformados para que, em seguida, métodos de redução de dimensionalidade e de aprendizado de máquina possam ser utilizados para a construção do modelo de recomendação de músicas. O resultado será um modelo híbrido capaz de gerar no *Spotify* uma playlist de músicas coerentes com as preferências do usuário, levando em consideração características individuais de cada faixa, como dançabilidade, energia ou volume, além das preferências musicais da pessoa com base em suas músicas mais ouvidas.

**Palavras-chave:** Sistema de Recomendação, Modelo Híbrido, Aprendizado de Máquina, Música .

# Lista de ilustrações

Figura 1 – Representação da estrutura de uma onda de som. Fonte: Adaptado de Miletto et al. (2004) . . . . .	12
Figura 2 – Formato de ondas simples e seus timbres. Fonte: Adaptado de Miletto et al. (2004) . . . . .	13
Figura 3 – Estrutura de uma árvore. Fonte: Do Autor . . . . .	17
Figura 4 – Distância entre (4,3) e os demais pontos. Fonte: Do Autor . . . . .	20
Figura 5 – Estrutura de árvore e seccção no espaço de uma Ball Tree. Fonte Adaptada de (OMOHUNDRO, 1989) . . . . .	21
Figura 6 – Árvore KD Tree. Fonte: Adaptado de (WICKMEDIA, 2024a) . . . . .	23
Figura 7 – Segmentação no espaço de 2 dimensões. Fonte: Adaptado de (WICKMEDIA, 2024b) . . . . .	24
Figura 8 – Variância por dimensão. Fonte: Do autor . . . . .	25
Figura 9 – Conjunto 'Iris' em $R$ sem e com $PCA$ . Fonte: Do Autor . . . . .	26
Figura 10 – Implementação do $t-SNE$ no conjunto MNist Fonte: Adaptado de (STELLING, 2022) . . . . .	26
Figura 11 – <i>Music Playlists Generator</i> Fonte: Adaptado de (ALMEIDA et al., 2017) . . . . .	29
Figura 12 – Exemplo de aplicação no site do <i>Spotify</i> . Fonte: Do autor . . . . .	31
Figura 13 – Flow da mineração com a <i>API</i> do <i>Spotify</i> . Fonte: Do autor . . . . .	32
Figura 14 – Estrutura relacional dos dados. Fonte: Do autor . . . . .	33
Figura 15 – Flow do treinamento com os dados. Fonte: Do autor . . . . .	34
Figura 16 – Comportamento dos dados por duas dimensões independentes. Fonte: Do Autor . . . . .	36
Figura 17 – $t-SNE$ plot de $R^{11}$ em $R^2$ . Fonte: Do autor . . . . .	36
Figura 18 – Clusterização com 3 gêneros. Fonte: Do autor . . . . .	37
Figura 19 – Variância por número de dimensões no sistema.Fonte: Do autor . . . . .	38
Figura 20 – Fluxo do contato do cliente. Fonte: Do autor . . . . .	39
Figura 21 – Página <i>Web</i> local. Fonte: Do autor . . . . .	41
Figura 22 – Prduto gerado na conta do usuário. Fonte: Do autor . . . . .	42
Figura 23 – Possível arquitetura para implementação futura. Fonte: Do autor . . . . .	47

# Lista de abreviaturas e siglas

HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
API	Application Programming Interface
PCA	Principal Component Analysis
KNN	K-Nearest Neighbors
AWS	Amazon Web Services
CSV	Comma-separated values
RDS	Relational Database Service



# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>9</b>
<b>1.1</b>	<b>Objetivos</b>	<b>9</b>
1.1.1	Objetivo Geral	9
1.1.2	Objetivos Específicos	10
<b>1.2</b>	<b>Organização do Trabalho</b>	<b>10</b>
<b>2</b>	<b>REVISÃO BIBLIOGRÁFICA</b>	<b>11</b>
<b>2.1</b>	<b>Fundamentação Teórica</b>	<b>11</b>
2.1.1	Revisão Sobre Conceitos Musicais	11
2.1.2	Conceitos Básicos do Som	11
2.1.3	Terminologia Musical	12
2.1.4	Sistemas de Recomendação	13
2.1.5	Algoritmos de Aprendizado de Máquina	15
2.1.5.1	Algoritmos baseados em Árvores	16
2.1.5.2	Técnicas dos vizinhos mais próximos	17
2.1.5.2.1	Técnicas sem estruturas <i>NN</i>	18
2.1.5.2.2	Técnicas com estruturas <i>NN</i>	20
2.1.5.2.3	<i>Ball Tree</i>	21
2.1.5.2.4	<i>KD Tree</i>	22
2.1.6	Algoritmos de Redução de Dimensionalidade	23
2.1.6.1	<i>Principal Component Analysis</i>	24
2.1.6.2	t-SNE	25
2.1.7	Ferramentas Arquiteturais	27
<b>2.2</b>	<b>Trabalhos Correlatos</b>	<b>27</b>
<b>3</b>	<b>DESENVOLVIMENTO</b>	<b>30</b>
<b>3.1</b>	<b>Mineração do conjunto de dados</b>	<b>30</b>
<b>3.2</b>	<b>Treinamento com os dados</b>	<b>34</b>
<b>3.3</b>	<b>Acesso ao perfil e criação da playlist</b>	<b>39</b>
<b>4</b>	<b>RESULTADOS</b>	<b>42</b>
<b>5</b>	<b>CONCLUSÃO</b>	<b>44</b>
	<b>REFERÊNCIAS</b>	<b>48</b>

# 1 Introdução

O desenvolvimento da Internet, dos dispositivos móveis e de diversas formas de entretenimento juntamente com a grande disseminação desses recursos entre as pessoas, causaram um impacto substancial no avanço das formas de lazer já estabelecidas e na criação de novas, levando a um aumento de conteúdos inéditos, novos temas e estilos. Na música, pode-se observar uma notável ascensão e popularização dos serviços de *streaming* de áudio. Estes serviços se tornaram uma ferramenta cotidiana para lazer, cultura e aprendizado, oferecendo conteúdos musicais de vários artistas, lugares e épocas de forma a atender públicos variados de forma rápida e eficaz.

Diferentes gostos, idade e personalidade acabam por modificarem as escolhas musicais de um indivíduo e, por consequência, há a necessidade de sistemas que consigam acompanhar a evolução dos conteúdos midiáticos e recomendar assuntos que possam encaixar-se ao público, à época e aos diversos tipos musicais que nascem em diferentes lugares do mundo. A expressão musical, uma parte essencial da experiência humana, pode ser encontrada desde a pré-história e, atualmente, é utilizada até para o tratamento de algumas doenças, como argumenta Michael H. Thaut (2015) em seu artigo.

Por conta disso, muitos autores têm se empenhado em construir e desenvolver algoritmos que possibilitem uma experiência eficiente para os usuários, como os trabalhos de Sá (2009), Monteiro et al. (2009) e Aliaga et al. (2019), que promovem a criação de sistemas capazes de comportar a grande quantidade de dados e usuários em ambientes de música. Ademais, há outros artigos que explicitam as dificuldades, possíveis caminhos e estratégias na criação deste tipo de modelo, como as *surveys* de Sá (2009) e Ko et al. (2022). Entretanto, estes trabalhos se concentram exclusivamente na teoria de um modelo de recomendação, sem efetivamente implementá-lo.

Ademais, outros autores elaboram um modelo híbrido de recomendação em seus trabalhos por meio de conteúdos públicos e camadas de armazenamento gerando, inclusive, um produto (GODINHO; VASCONCELOS, ). Entretanto, a teoria estatística por trás destes modelos não é muito explorada.

## 1.1 Objetivos

### 1.1.1 Objetivo Geral

Esta monografia busca construir um sistema de recomendação híbrido compreendendo as dificuldades e caminhando por possíveis soluções discutidas pelos artigos acima, ou seja, unindo a teoria juntamente com o ferramental para criar um produto. Além

disso, estudando os métodos matemáticos necessários para o tratamento dos dados e treinamento do algoritmo, assim como explicitando os problemas encontrados no meio do caminho, obtendo ao final do trabalho uma playlist no perfil do usuário.

### 1.1.2 Objetivos Específicos

1. Autenticar com a *API* do *Spotify*.
2. Raspar a *API* para obtenção das músicas de todos os gêneros utilizando a linguagem *Python*.
3. Armazenar os dados em um Banco de Dados.
4. Limpar e visualizar as informações relevantes no conjunto de dados utilizando a linguagem *R*.
5. Utilizar técnicas de redução de dimensionalidade para tornar o modelo mais eficiente.
6. Treinar os dados da etapa anterior sobre o algoritmo de *KNN*.
7. Desenvolver um servidor simples local para chamadas de *API*.
8. Obter os dados do usuário no *Spotify*.
9. Treinar os dados do item 8 com o modelo treinado no item 6.
10. Gerar o conjunto de músicas respostas do item 9 no perfil do usuário usando o servidor desenvolvido no item 7.

## 1.2 Organização do Trabalho

Nesta monografia, a Introdução apresenta um resumo do tema, explica as motivações para a criação deste trabalho e descreve como ele foi desenvolvido e estruturado. Na Revisão Bibliográfica, termos musicais, conceitos matemáticos e questões arquiteturais serão discutidos de forma independente, visando facilitar a compreensão do leitor. Estes conceitos serão posteriormente retomados nos capítulos seguintes, onde serão empregados de forma correlacionada na construção do modelo híbrido. O capítulo Desenvolvimento trata das nuances do trabalho, explorando os caminhos tomados e apresentando os problemas encontrados para geração do produto final. Uma análise detalhada sobre o produto final e sua qualidade é feita no capítulo de Resultados. Por fim, a Conclusão retoma a discussão sobre o sistema construído assim como as limitações encontradas e os possíveis caminhos para variações futuras desta monografia.

## 2 Revisão Bibliográfica

Neste capítulo, serão explorados temas fundamentais para a construção deste trabalho, visando proporcionar uma compreensão mais profunda dos tópicos abordados e dos trabalhos correlatos utilizados no desenvolvimento.

### 2.1 Fundamentação Teórica

#### 2.1.1 Revisão Sobre Conceitos Musicais

A Computação Musical é um dos ramos da Ciência da Computação assim como a Computação Gráfica, só que essa ao invés de analisar conceitos de visualização e modelagem, propõe-se a estudar os atributos básicos da área de música, suas aplicações, algoritmos de conversão e compartilhamento desse tipo de conteúdo ([MILETTO et al., 2004](#)).

#### 2.1.2 Conceitos Básicos do Som

Desde o momento do nascimento até a morte, o ser humano está rodeado de sons, muitos dos quais são desconhecidos e novos para ele. À medida que ele cresce, ele passa a compreender muitas das características do som. No entanto, as estruturas básicas podem passar despercebidas. Por isso, estudos como os de [Miletto et al. \(2004\)](#) e [Orio et al. \(2006\)](#) exploram questões como “O que é som?” e “O que é música?”, na tentativa de esclarecer seus componentes.

Ao tocar uma música no carro uma sequência de *bits* é lida por algum programa que converterá esse *bits* em sinais elétricos. O sinais elétricos serão enviados para as caixas de som que, por sua vez, comandarão os altos falantes a movimentar o ar com uma força e velocidade controladas; esse movimento é realizado de maneira repetitiva, resultando em deformações e restaurações periódicas chamadas de vibração. Essas vibrações serão captadas pelas terminações nervosas que darão início ao processo conhecido como audição. Assim, sons são vibrações causados pelo deslocamento de massas de ar por algum objeto. No espaço, por exemplo, não há ar e portanto não há som.

O som possui 3 grandes elementos básicos sendo eles a altura tonal, o volume e o timbre ([MILETTO et al., 2004](#)).

- **Altura Tonal** pode ser percebida em instrumentos como piano ou teclado, pois ao tocar nas notas mais à esquerda o som será mais grave, enquanto que ao tocar nas

notas mais à direita o som será mais agudo. Como mencionado anteriormente, as vibrações são periódicas e portanto formam ciclos (Figura 1) que podem ser medidos pelo tempo até sua repetição. O número de ciclos dentro do intervalo de um segundo é geralmente chamado de frequência e expresso em unidades chamadas *Hertz* (*Hz*) (MILETTO et al., 2004). Quanto maior o valor em *Hz* mais agudo será o som, e quanto menor, mais grave.

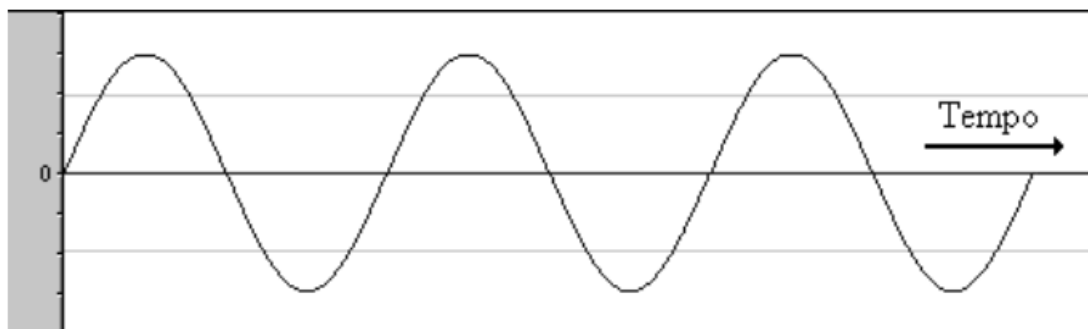


Figura 1 – Representação da estrutura de uma onda de som. Fonte: Adaptado de Miletto et al. (2004)

- **Volume** pode ser observado quando a tecla do teclado for pressionado com mais força gerando assim um som mais forte, enquanto que se for tocado fracamente, o som será fraco. O conceito de alto e baixo deve ser utilizado de maneira certa referindo-se apenas a altura tonal e não ao volume, já que a mudança no volume do som ocorre por variação da amplitude de onda. Em resumo, o volume é determinado pela altura da onda (amplitude) e seu atributo classificado muitas vezes como alto ou baixo é errado, sendo melhor explicado como forte ou fraco diante da sua amplitude de onda (MILETTO et al., 2004).
- **Timbre** pode ser observado ao tocar dois instrumentos usando a mesma altura tonal e volume, já que ao ser tocado nessas condições ambos os instrumentos terão diferentes sons gerados. Isso acontece por conta do seu formato de onda, ondas mais arredondadas possuem um timbre mais suave enquanto que ondas mais pontiagudas possuem um timbre mais penetrante e estridente (Figura 2).

### 2.1.3 Terminologia Musical

Alguns outros conceitos são importantes de serem lembrados, ou ainda de serem apresentados para pessoas que possuem um conhecimento musical superficial. Estes conceitos estão presentes na teoria musical e são componentes essenciais da música, sendo fundamentais tanto na sua criação quanto na sua classificação.

Estes termos são a tonalidade, o *tempo*, a *time-signature* e a *key signature* como argumenta Orio et al. (2006).




Forma de onda	Timbre	Instrumento
<i>Onda senoidal</i> 	<i>suave</i>	<i>flauta, assovio</i>
<i>Onda dente de serra</i> 	<i>claro</i>	<i>violino, trompete</i>
<i>Onda retangular</i> 	<i>simples</i>	<i>clarinete, oboé</i>

Figura 2 – Formato de ondas simples e seus timbres. Fonte: Adaptado de [Miletto et al. \(2004\)](#)

- A **tonalidade** de um som está relacionada ao papel de diferentes acordes em uma obra musical. A tonalidade é definida pelo acorde principal a qual a música segue e pode não ser aplicado a certos gêneros.
- O **tempo** é a velocidade na qual a música é tocada ou esperada ser tocada pelos músicos; é normalmente medido por *beats per minute* (*bpm*).
- O **time-signature**, normalmente denotado na forma de um número fracionário, dá a informação da organização entre batidas fortes e suaves através do eixo do tempo.
- O **key signature**, normalmente mostrado na forma de números de alterações, é uma forma incompleta da tonalidade e muito usado pelos músicos para saber onde as notas se alteram.

#### 2.1.4 Sistemas de Recomendação

Sistemas de recomendação são ferramentas construídas para ajudar o consumidor na descoberta de novos conteúdos, produtos ou serviços, ao agregar e analisar sugestões de outros usuários ou produtos previamente avaliados ([PARK et al., 2012](#)). Estes tipo de sistema tornou-se cada vez mais relevante e importante em decorrência da propagação de dispositivos tecnológicos assim como o surgimento de aplicativos que possibilitam facilmente o acesso a diversas formas de entretenimento.

A música é uma parte essencial do cotidiano de muitas pessoas ao redor do mundo, sendo a atividade mais frequente do que todas as outras como mostra pesquisas anteriores ([SONG; DIXON; PEARCE, 2012](#)), e uma parte essencial da cultura humana podendo ser usada até mesmo para cura de doenças. Por conta disso, sistemas de recomendação que sejam robustos e eficientes para conseguir dar uma resposta rápida e acurada diante do grande conjunto de possibilidades presentes fazem-se necessários.

Um sistema de recomendação ideal conseguiria sugerir músicas personalizadas para diferentes usuários da melhor maneira possível. Em termos formais, considere que  $C$  seja o conjunto de todos os usuários e que  $S$  seja o conjunto de todas as possíveis recomendações. Note que o espaço  $S$  pode conter milhares ou milhões de itens possíveis (músicas, livros ou filmes). Seja  $u$  uma função de utilidade que mensura a utilidade de um item  $s$  para um usuário  $c$ . Procuramos no espaço de possibilidades para cada usuário  $c$  um item  $s$  que maximizará a função  $u$ , isto é, procuramos o valor máximo  $s'_c$  de  $u$  que será a melhor recomendação possível (ADOMAVICIUS; TUZHILIN, 2005):

$$\forall c \in C, \quad s'_c = \operatorname{argmax}_{s \in S} u(c, s) \quad (2.1)$$

Diversos tipos de abordagem para o mesmo problema foram desenvolvidos. As técnicas mais comuns são: os modelos baseados em conteúdo, os filtros colaborativos e os híbridos. Os modelos baseados em conteúdo fundamentam-se na capacidade do sistema de reconhecer com base em assuntos previamente classificados pelo usuário, potenciais novos conteúdos que possam ser do seu interesse. Esse modelo será usado nesse trabalho por meio da *API* do *Spotify* que fornecerá as faixas mais ouvidas do usuário.

Os filtros colaborativos são semelhantes ao estilo de recomendação mais tradicional, o “boca a boca”, no qual pessoas próximas a você recomendam músicas (produtos) que elas ouviram e gostaram, ou seja, por meio da classificação de terceiros com alta proximidade do usuário alvo, o sistema identifica padrões e sugere materiais frequentemente consumidos pela comunidade próxima (vizinhança).

O modelo híbrido é uma mistura de um ou mais sistemas de recomendação, onde a fraqueza de um modelo pode ser atenuada ou removida quando mesclado com um outro sistema (ADOMAVICIUS; TUZHILIN, 2005). Sua construção é um dos objetivos deste trabalho, por meio do uso de outros dois tipos de sistemas juntos.

Com o passar do tempo outras novas abordagens surgiram na tentativa de superar as limitações das anteriores, entre elas destaca-se a recuperação de informação por metadados. Esta abordagem utiliza características fundamentais do som, como timbre, volume e características acústicas, para recomendar sons similares, sendo este um dos modelos escolhidos para implementação na monografia.

Além das abordagens mencionadas anteriormente, o modelo baseado em contexto propõe-se a analisar informações sobre o momento do dia, a atividade que será realizada enquanto consumindo o material recomendado ou a finalidade do conteúdo. Uma playlist para ouvir na academia ou dirigindo certamente seriam diferentes em estilos e características musicais e assim, por meio do contexto, busca-se prever o tipo de músicas que melhor adequam-se a situação.

Por final, os modelos baseados em emoção, utilizam do sentimento do usuário

para conseguir prescrever uma série de conteúdos que melhor encaixariam ao humor dele, buscando criar um sentimento de empatia ou até mesmo de conforto para o ouvinte (SONG; DIXON; PEARCE, 2012).

### 2.1.5 Algoritmos de Aprendizado de Máquina

Desde que os computadores foram criados, conjecturou-se da possibilidade deles serem capazes de pensar, construir e se expressar de forma autônoma. Para sustentar esse desejo, muitos caminhos tiveram que ser pavimentados.

Historicamente, o termo aprendizado de máquina (*machine learning*) tem origem dos trabalhos do psicólogo Frank Rosenblatt da Universidade Cornell, que baseou seus estudos sobre o sistema nervoso humano para construir um sistema que reconhecesse o alfabeto de Rosenblatt. Este sistema foi chamado de 'Perceptron' e se tornou o protótipo das redes neurais artificiais. Seu desenvolvimento assemelhou-se ao aprendizado observado de animais e seres humanos na psicologia. Contudo, algumas limitações intrínsecas à sua implementação o impediam de atingir todos os objetivos pretendidos, como exemplo, o problema do XOR.

Por conta da criação dessa nova técnica que representava o cumprimento de um antigo sonho, muitos pesquisadores ficaram entusiasmados e propuseram-se a tentar resolver essas limitações bem como melhorá-la ou desenvolvê-la em campos antes inexplorados. Com o desenvolvimento nessa área, as grandes empresas perceberam que as técnicas desenvolvidas poderiam resolver problemas do mercado. Isso resultou em uma onda massiva de investimentos na esperança de encontrar soluções para antigos desafios do mercado (FRADKOV, 2020).

Entretanto, a falta de poder computacional, de disponibilidade de dados e hardware eficiente para essas técnicas serem implementadas e desenvolvidas gerou um desânimo nas instituições e empresas: muitos investimentos para poucos retornos. Em decorrência disso, iniciou-se um período de desaceleração na área de aprendizado de máquina que passou a ser chamado na História de *Cold Winter*. Seu renascimento começou no início do século XXI causado pelo início da era de *BigData*, assim como uma explosão no poder computacional marcado pelo paralelismo de processamento, como nas *GPU's* da NVIDIA, e pela acessibilidade do hardware dada pelo seu barateamento em mercado. Este período tem sido chamado de *The Gold Rush of Machine Learning* (FRADKOV, 2020).

Uma das características fundamentais do *BigData* é a grande quantidade de dados disponíveis, muitos dos quais são e acessíveis para análises. Entretanto, muitas das informações presentes na Internet não são bem estruturadas e seu tratamento muitas vezes é inviável ou demandará muito tempo dentro de todo o processo de aprendizado



de máquina. Em geral, são duas as principais técnicas de aprendizado de máquina: o aprendizado não supervisionado e o aprendizado supervisionado.

As técnicas de aprendizado não supervisionado tem sido usadas muito nos últimos anos e sua principal característica é que o conjunto de dados que será submetido a esta técnica não necessita de uma variável resposta. Nos modelos não supervisionados, estamos mais interessados em encontrar padrões e semelhanças entre as observações do conjunto. Assim, seus objetivos podem variar dependendo do problema, indo desde a visualização de dados em altas dimensões passando por reconhecimento de imagens e chegando no agrupamento de clientes de uma empresa. Como no modelo não supervisionado as observações não possuem uma classificação prévia, este modelo busca, então, estabelecer padrões entre as observações. Em geral, a busca pelos padrões entre as observações se dá a partir de técnicas de aglomeração (usualmente chamadas na literatura de *clustering*), como no modelo de aglomeração hierárquica e no modelo de k-médias. Em muitos casos, estes problemas esbarram em vários desafios, como a grande quantidade de variáveis envolvidas.

No aprendizado supervisionado existe uma variável resposta  $Y$  (também chamada de variável dependente) que será modelada a partir do vetor  $\vec{X}$  de variáveis explanatórias (independentes) do conjunto de dados. As técnicas do aprendizado supervisionado podem ser divididas em duas categorias: regressão e classificação (SOOFI; AWAN, 2017). Na regressão, a variável resposta  $Y$  assume valores numéricos contínuos; já na classificação, a variável  $Y$  é do tipo categórica.

#### 2.1.5.1 Algoritmos baseados em Árvores

Algoritmos baseados em árvores podem ser usados tanto para classificação como para regressão, envolvendo processos de estratificação e segmentação no conjunto de dados. Esses métodos são simples e úteis para interpretação, sendo usados em algoritmos como Árvores de Decisão, Florestas Aleatórias e Árvores de Regressão Aditivas Bayesianas (JAMES DANIELA WITTEN; TIBSHIRANI, 2013).

Na estrutura de uma árvore há 3 tipos de vértices essenciais.

- Nó raiz: é o primeiro nó da árvore sendo o único que não possui pais, mas pode possuir filhos; este tipo de nó está representado na Figura 3 pelo nó **A**.
- Nó intermediário (ou nó interno): são os nós que possuem pais e filhos; este tipo de nó está representado pelo nó **B** na Figura 3.
- Nó folha: são os nós que somente possuem pais e não possuem filhos; representados pelos nós **C**, **D** e **E** na Figura 3.

As árvores de decisão podem ser construídas levando-se em consideração diferentes tipos de algoritmos. Nas árvores do tipo *Binary Trees*, um nó pai tem até dois filhos, mas também existem árvores onde os nós tem 3 filhos (um à esquerda, outro ao meio e outro à direita) ou até mesmo árvores que possuem  $n$  filhos (sendo chamadas de *Generic Tree*). Cada uma destas árvores possui suas variações com balanceamento e degeneração e cada qual tem a sua abstração para os nós da estrutura principal.

Nessa monografia, o uso de árvores será usada como estrutura para divisão do conjunto de dados em altas dimensões ( $\mathbb{R}^n$ , em que  $n > 2$ ) de forma a facilitar o treino do modelo escolhido. As técnicas com estrutura de árvores testadas no conjunto extraído do *Spotify* neste trabalho são chamadas de *KD Tree* e de *Ball Tree* e serão explicadas posteriormente.

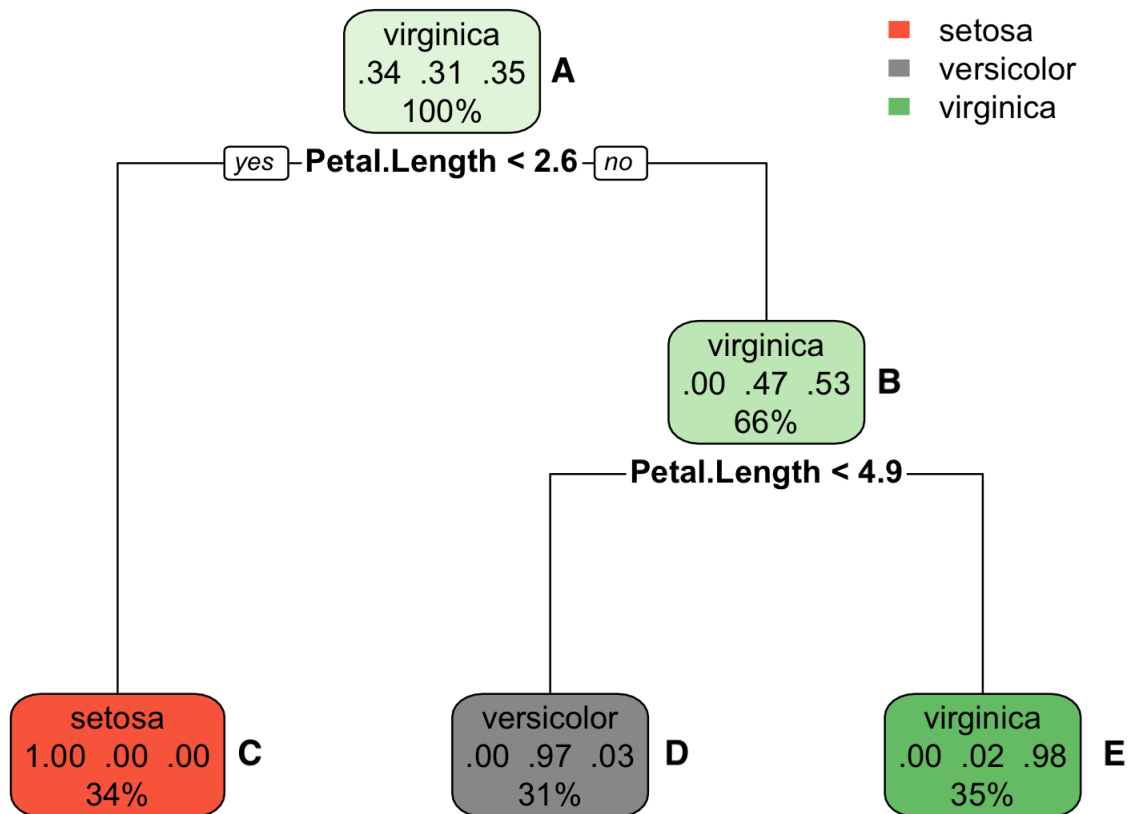


Figura 3 – Estrutura de uma árvore. Fonte: Do Autor

#### 2.1.5.2 Técnicas dos vizinhos mais próximos

O *K-Nearest Neighbors* (*KNN*) é um algoritmo de aprendizado de máquina que busca identificar uma classe desconhecida de um ponto por meio de seus vizinhos mais próximos cuja a classe desejada é conhecida. Normalmente o termo 'mais semelhante' a uma observação fixa é dado pelo ponto mais próximo a esta observação. Em muitos casos,

a distância considerada é a distância euclidiana, mas não necessariamente isso é uma regra, sendo que outras distâncias como a de Manhattan ou a distância do cosseno podem ser muito úteis dependendo do problema em questão.

A complexidade desse algoritmo é  $O(N^2)$  sendo difícil de calcular para valores de  $N$  grandes. Por conta desta limitação uma variação com pesos foi criada. Nela os dados de treino são assimilados com pesos juntamente com a sua distância dos pontos, contudo o tempo computacional e o uso da memória ainda são um grande problema a serem resolvidos. Para tentar amenizar ainda mais esses empecilhos, alguns pesquisadores usam variações do algoritmo que descartam pontos que tenham padrões repetidos ou até mesmo pontos que não afetarão em nada o conjunto de treino (BHATIA et al., 2010).

Um fator importante para o bom desempenho do algoritmo é o número de  $K$  vizinhos considerados para se avaliar a vizinhança de um ponto. Algumas técnicas, como o *cross-validation*, podem ser usadas para tentar achar um  $K$  ótimo.

Na monografia, a técnica de *KNN* foi usada inicialmente para recomendar músicas com atributos musicais semelhantes, como volume, energia e outros valores que serão discutidos mais a frente, em um conjunto de dados de quase 40.000 músicas. Contudo, muitos problemas foram encontrados, sendo eles:

- Como aumentar a eficiência de tempo de um algoritmo que precisa computar a distância euclidiana entre pares de 40.000 músicas ?
- Com a inserção de mais uma música, linhas e colunas dessa matrix devem ser reprocessadas. Como reduzir esse custo?
- Semelhante ao anterior, o processo de exclusão também oferece os mesmos custos, como reduzi-los ?

Para isso, outras técnicas de *Nearest Neighbors (NN)* onde o conjunto do espaço de possibilidades é seccionado de maneiras estratégicas foram estudadas. O trabalho iniciou-se com uma abordagem usando o *KNN* e depois foram testadas outras duas técnicas, o *KD Tree* e o *Ball Tree*, ambas tendo como base a estrutura de árvores para aperfeiçoar a construção, inserção e deleção.

#### 2.1.5.2.1 Técnicas sem estruturas *NN*

Para tentar eliminar algumas das limitações de tempo computacional na sua construção e treinamento, variações do *Nearest Neighbor (NN)* sem estrutura foram criados, como o *Weighted KNN (WKNN)*, onde, além de se ordenar os pontos pela distância, é também considerado o peso entre um ponto e outro (BHATIA et al., 2010). Além disso,

para tentar economizar espaço na memória, podem ser consideradas variações como *Condensed Nearest Neighbor (CNN)* ou *Reduced Nearest Neighbor (RNN)*, onde o algoritmo armazena os padrões dos dados e elimina padrões repetidos ou que não acrescentam informações ao resultado do treinamento como descreve [Bhatia et al. \(2010\)](#) em sua *survey*.

O *KNN* está nessa categoria porque ele é o processamento entre um determinado ponto para com os outros  $N - 1$  pontos sem nenhuma estrutura de dados que o sustente, resultando em uma complexidade  $O(N^2)$  e um matrix sensível a mudanças, necessitando de reproprocessamento a cada inserção ou exclusão. Para sua construção, deve haver um conjunto de dados com  $I$  elementos e com  $N$  atributos. Para cada ponto em  $I$ , sejam  $i, j \in \mathbb{N}$  e sejam  $p_i, p_j \in I$ . Denotando  $p_i = (P_{i1}, P_{i2}, \dots, P_{iN})$  e  $p_j = (P_{j1}, P_{j2}, \dots, P_{jN})$ , a distância euclidiana entre  $p_i$  e  $p_j$  será denotada por  $D_{(i,j)}$  e será dada por:

$$D_{(i,j)} = \sqrt{\sum_{k=1}^N (P_{ik} - P_{jk})^2} \quad (2.2)$$

A partir destas distâncias, podemos construir a matriz  $D$  de distâncias em que a entrada  $(i, j)$  desta matriz será dada por  $D_{(i,j)}$ . Logo, a matriz  $D$  será simétrica e terá sua diagonal principal igual a zero, pois  $D_{(i,i)} = 0$ ,  $i = 1, 2, 3, \dots, N$ . A seguir, uma exemplificação desta matriz de distâncias:

$$\begin{vmatrix} 0 & D_{(0,1)} & D_{(0,2)} & \dots & D_{(0,j)} \\ D_{(1,0)} & 0 & D_{(1,2)} & \dots & D_{(1,j)} \\ D_{(2,0)} & D_{(2,1)} & 0 & \dots & D_{(2,j)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{(i,0)} & D_{(i,1)} & D_{(i,2)} & \dots & 0 \end{vmatrix}$$

Como um exemplo, dado os pontos  $(4, 3), (1, 3), (5, 2), (8, 1), (2, 7), (5, 1), (6, 2), (3, 6), (4, 3)$ , a primeira linha será preenchida pelas distâncias entre o ponto  $(4, 3)$  e as demais existentes, como mostrado na [Figura 4](#).

Ao fazer o processamento das distâncias dos  $N$  pontos, apresentamos logo após a [Figura 4](#) a matrix de distâncias resultante do conjunto.

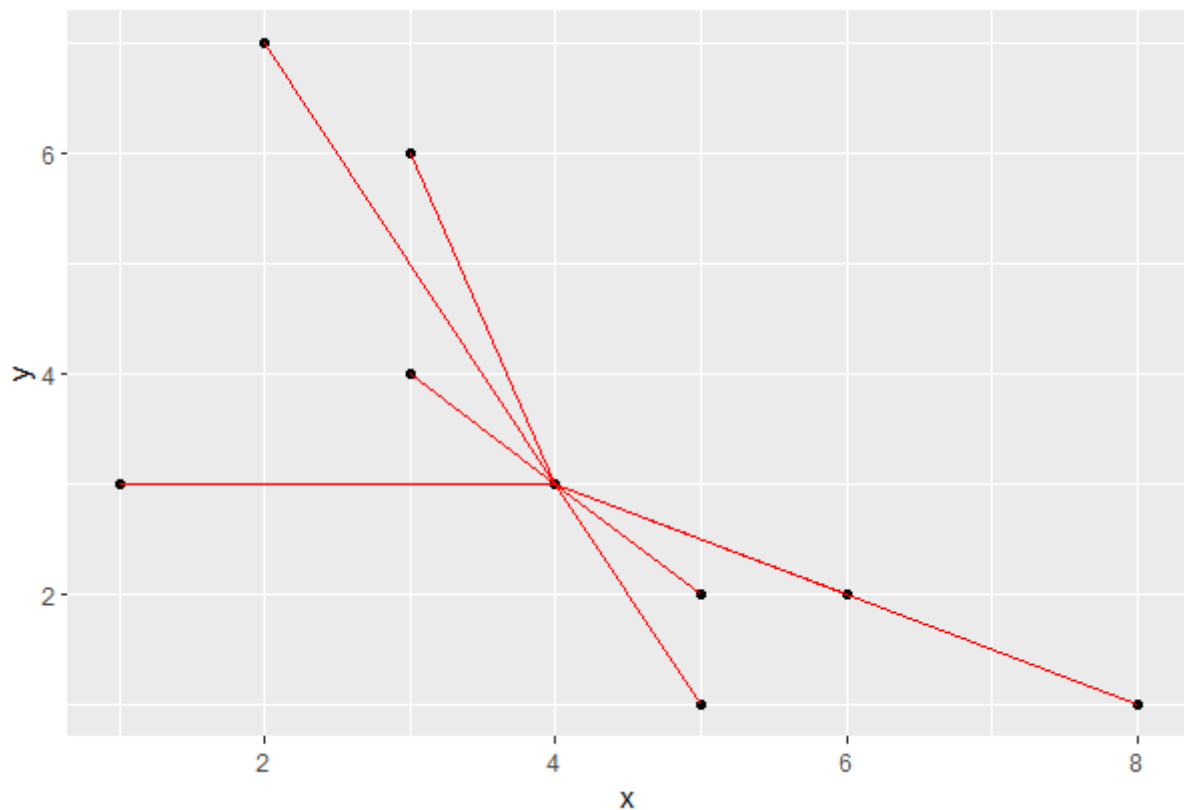


Figura 4 – Distância entre (4,3) e os demais pontos. Fonte: Do Autor

0	1.41	1.41	4.47	4.47	2.23	2.23	3.16	3.00
1.41	0	2.82	5.83	3.16	3.60	3.60	2.00	2.36
1.41	2.82	0	3.16	5.83	1.00	1.00	4.47	4.12
4.47	5.83	3.16	0	8.48	3.00	2.23	7.07	7.28
4.47	3.16	5.83	8.48	0	6.70	6.40	1.41	4.12
2.23	3.60	1.00	3.00	6.70	0	1.41	5.38	4.47
2.23	3.60	1.00	2.23	6.40	1.41	0	5.00	5.09
3.16	2.00	4.47	7.07	1.41	5.38	5.00	0	3.60
3.00	2.36	4.12	7.28	4.12	4.47	5.09	3.60	0

Para o algoritmo *KNN*, os *K*-vizinhos de um ponto *X* serão os *K* pontos correspondentes aos *k* menores da linha (ou coluna) *X* com exceção do próprio elemento *X*. Ainda sim, a inserção ou deleção de um elemento mudará  $N - 1$  valores nas linhas e  $N - 1$  nas colunas o que mostra seu problema de acomodação de mudanças.

#### 2.1.5.2.2 Técnicas com estruturas *NN*

Nessa categoria são propostas as variações do algoritmo de *NN* com o uso de estruturas desde árvores até algoritmos como o *Principal Component Analysis (PCA)*. O

objetivo é facilitar o treinamento de modelo e melhorar a acurácia e eficiência computacional.

### 2.1.5.2.3 *Ball Tree*

Esse algoritmo baseado em árvores binárias é uma estrutura de dados que particiona o espaço de forma a organizar os pontos em planos multidimensionais, usando-se de um conjunto aninhado de círculos para divisão destas observações.

Formalmente, cada partição é feita recursivamente de modo a cortar o espaço em hiperesferas. Cada nó, que não é um nó-folha, é uma dessas hiperesferas contendo um subconjunto do seu nó pai e os nó-folhas são o menor subconjunto de pontos para aquele conjunto de dados onde é carregado as informações necessárias para a aplicação. Vale ressaltar que o processo de divisão é feito por um “pivô” escolhido dentro do subconjunto de dados e as esferas são construídas por meio deste ponto escolhido.

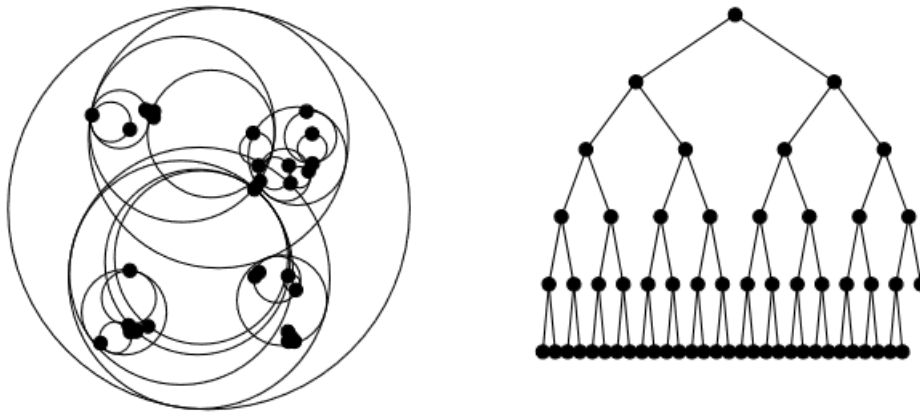


Figura 5 – Estrutura de árvore e seccção no espaço de uma Ball Tree. Fonte Adaptada de (OMOHUNDRO, 1989)

**Algoritmo 1:** Construção de uma *Ball Tree***Entrada:** Conjunto de dados de  $\mathbb{R}^n$  dimensões**Saída:** Raiz da árvore *Ball Tree*


---

```

1 se há somente um ponto então
2   |   Crie um nó folha com esse ponto na árvore;
3   |   retorna nó raiz da árvore;
4 senão
5   |   Calcular dentre as  $N$  dimensões, a dimensão  $I$  que possui maior variância;
6   |   Achar na dimensão  $I$ , a mediana dos pontos chamará de pivô;
7   |   Crie dois círculos a partir do pivô;
8   |   Pontos mais pertos do centro do primeiro círculo vão para o conjunto E;
9   |   Pontos mais pertos do centro do segundo círculo vão para o conjunto D;
10  |   Adicione o pivô como nó na árvore;
11  |   Chama função novamente para E;
12  |   Chama função novamente para D;
13 fim

```

---

Usando o algoritmo de *Ball Tree*, a inserção de um novo valor será da ordem  $O(\log N)$  e a remoção também será  $O(\log N)$ , sendo eficiente para dados cujo comportamento é esparso. A árvore resultante é naturalmente balanceada e a quantidade de filhos para cada nó é arbitrário.

2.1.5.2.4 *KD Tree*

Uma árvore *KD* é um algoritmo baseado em árvore em que cada nó intermediário representa um corte em um hiperplano associado a uma das  $N$  dimensões do conjunto de dados. Essa segmentação é feita a partir de um pivô (normalmente a mediana dos pontos) que dividirá o espaço em dois, resultando em um nó na árvore à direita e outro à esquerda (OMOHUNDRO, 1989). Por exemplo, considere um conjunto de dados em 2 dimensões com as seguintes observações: (2, 3), (5, 4), (9, 6), (4, 7), (8, 1), (7, 2). A árvore *KD* associada a este conjunto está representada na Figura 6. Sua divisão no espaço de duas dimensões será com segmentações nos eixos  $x$  e  $y$  como mostrado na Figura 7.

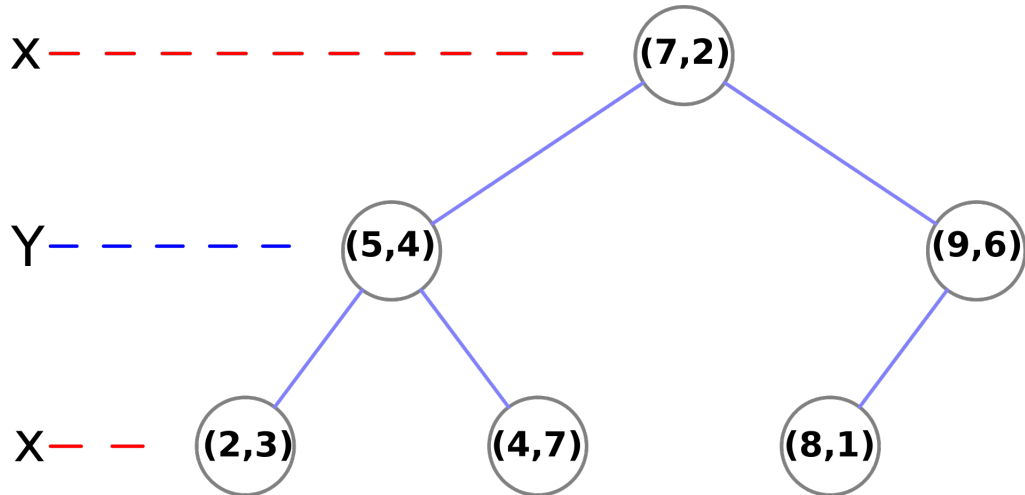


Figura 6 – Árvore KD Tree. Fonte: Adaptado de (WICKMEDIA, 2024a)

---

**Algoritmo 2:** Construção de uma *KD Tree*

---

**Entrada:** Conjunto de dados em  $\mathbb{R}^n$

**Saída:** Raiz da árvore *KD Tree*

```

1 se há somente um ponto então
2   Crie um nó folha com esse ponto na árvore;
3   retorna nó raiz da árvore;
4 senão
5   Calcular dentre as  $N$  dimensões, a dimensão  $I$  que possui maior variância;
6   Achar na dimensão  $I$ , a mediana dos pontos que chamaremos de pivô;
7   Crie dois conjuntos  $E$  e  $D$  repartindo o espaço em dois;
8   Adicione o pivô como nó na árvore;
9   Chame função novamente para  $E$ ;
10  Chame função novamente para  $D$ ;
11 fim

```

---

A sua estrutura geral difere-se do *Ball Tree* no momento em que sua árvore não é naturalmente balanceada e suas segmentações no espaço são normalmente retangulares e não circulares. Por outro lado, o número de filhos de cada nó é exatamente 2 e sua construção é mais rápida, ocupando, também, menos espaço na memória.

### 2.1.6 Algoritmos de Redução de Dimensionalidade

Na era de *Big Data*, conjunto de dados que possuem muitos atributos tornaram-se mais comuns, variando desde amostras de sangue até vídeos de alta resolução. Portanto, dados com dimensões grandes precisam ser tratadas de maneira computacionalmente eficiente em tempo e espaço para poderem ser usados no dia a dia.

Por conta disso, métodos matemático de redução de dimensionalidade ( $\mathbb{R}^m \Rightarrow \mathbb{R}^n$ ,



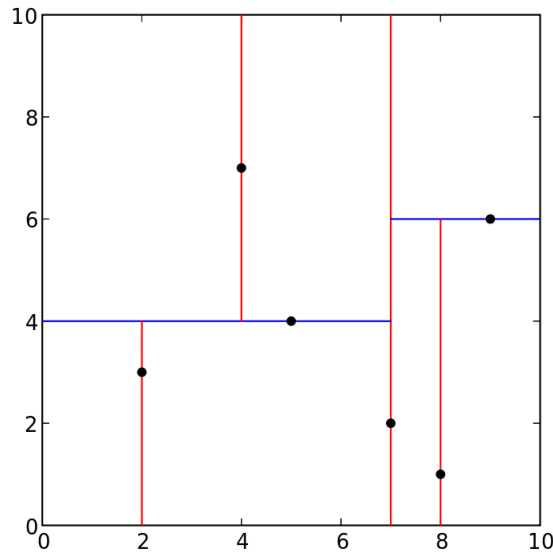


Figura 7 – Segmentação no espaço de 2 dimensões. Fonte: Adaptado de ([WICKMEDIA, 2024b](#))

em que  $m > n$  e  $m, n \in \mathbb{N}$ ) são necessários para conseguir reduzir o ruído no conjunto de dados, aumentar a precisão do algoritmo e melhorar a eficiência computacional no modelo que está sendo empregado.

Além disso, os métodos de redução de dimensionalidade não somente reduzem a complexidade do problema como podem, em alguns casos, aumentar a capacidade de interpretação da análise. Ademais, conjuntos de dados com dimensões menores possuem menor chance de *overfitting* para um mesmo modelo de aprendizado de máquina. Contudo, o grande desafio para se reduzir a dimensionalidade está em quantas dimensões podem ser removidas para que informação relevante não seja descartada.

Nessa monografia, o uso dessas técnicas foram usadas também para lidar com outra dificuldade conhecida na literatura como *curse of dimensionality*. Esse conceito foi primeiramente introduzido no livro *Dynamic Programming* de Richard Bellman ([BELL-MAN; CORPORATION; COLLECTION, 1957](#)). Neste livro, o autor argumenta que a quantidade de informação necessária para que haja uma generalização para treinamento dos dados cresce exponencialmente em relação a quantidade de atributos ou dimensões presentes, uma vez que os pontos começam a ficar esparsos no hiperespaço.

#### 2.1.6.1 Principal Component Analysis

*Principal Component Analysis* ou *PCA* é uma técnica que busca reduzir o número de dimensões de um conjunto sem perder muita informação. Ao se reduzir a dimensionalidade pode-se perder acurácia do modelo, mas por outro lado pode-se ganhar em simplicidade de interpretação.

Os componentes principais (PC) como são chamados, são novos valores criados por meio da combinação linear de valores anteriores (SINGER, 2013). Essas combinações são feitas de forma que cada nova variável é não correlacionada com as variáveis anteriores. A direção da primeiro componente principal do conjunto é a componente ao longo da qual as observações variam mais, como mostrado na Figura 8.

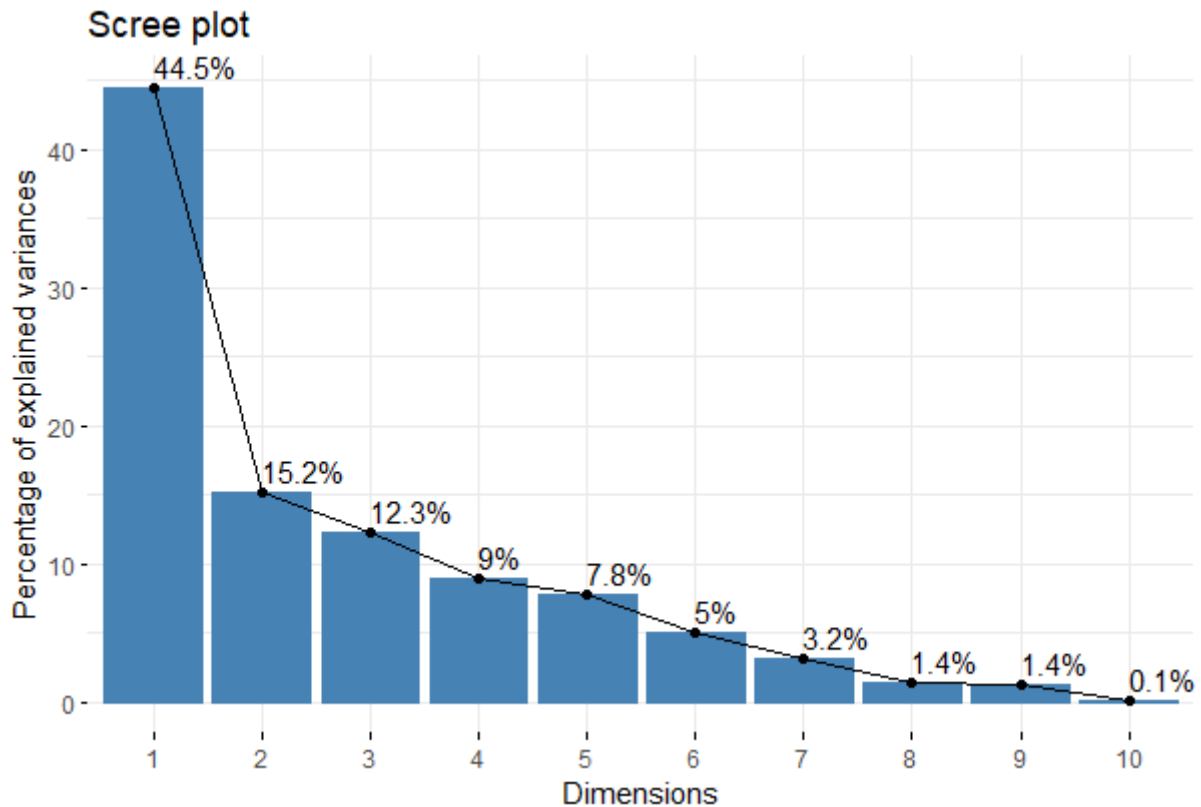


Figura 8 – Variância por dimensão. Fonte: Do autor

O gráfico da Figura 8 pode auxiliar na escolha do número de componentes principais que serão consideradas no processo de redução de dimensão: escolhe-se o menor número de componentes que são necessárias para explicar uma grande quantidade da variância dos dados. O uso do PCA destaca-se em conjuntos de dados de altas dimensões ( $\mathbb{R}^n$ ,  $n > 2$ ) de modo a aglutinar classes que tem uma quantidade alta de correlação em grupos, resultando em um aumento de acurácia no modelo final. Um exemplo do uso do algoritmo é mostrado na Figura 9, aonde as duas primeiras componentes principais conseguem explicar mais de 97% da variância do modelo.

#### 2.1.6.2 t-SNE

O *t-SNE* é uma técnica de visualização de dados em altas dimensões que converte a afinidade de pontos em probabilidades. Ele mantém o comportamento do espaço original por meio do mapeamento dos dados de probabilidades conjuntas em uma curva Gaussiana

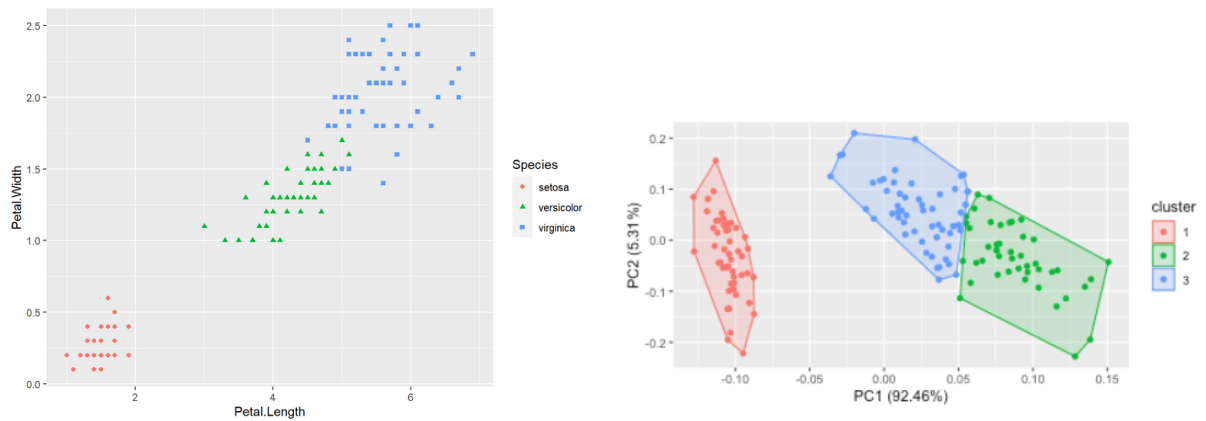


Figura 9 – Conjunto 'Iris' em  $R$  sem e com  $PCA$ . Fonte: Do Autor

em pontos mapeados para curva  $t$ -Student, o qual será usada para visualização ou criação de um novo espaço cartesiano como feito em Almeida et al. (2017).

Esse método não garante convergência, uma vez que conjuntos de dados não separáveis em altas dimensões não serão separados em baixas dimensões. Contudo, esta técnica ainda fornece visualizações que podem auxiliar na busca por informações sobre o comportamento do conjunto. A Figura 10 apresenta a visualização em duas dimensões, a partir do  $t$ -SNE, para um conjunto que possui 748 dimensões.

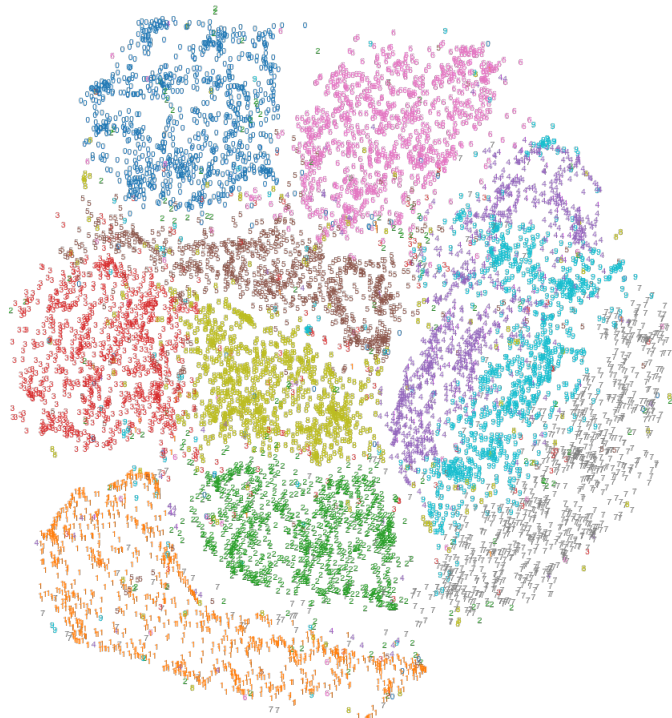


Figura 10 – Implementação do  $t$ -SNE no conjunto MNIST Fonte: Adaptado de (STELLING, 2022)

### 2.1.7 Ferramentas Arquiteturais

Nessa monografia, além dos tópicos matemáticos, algumas ferramentas computacionais serão necessárias para a criação do produto final desse trabalho. As linguagens *R*<sup>1</sup> e *Python*<sup>2</sup>, nas versões 4.3.1 e 3.10.11 respectivamente, serão usadas para *scraping* e manipulação dos e, por fim, para a implementação do algoritmo de *machine learning*. Nas duas linguagens se padronizou a *seed* no valor **1002**.

Além disso, será usado um *back-end* e *front-end* local, o primeiro sendo construído com *fast-api*<sup>3</sup> para que haja a comunicação através dos métodos *HTTP* (*Hypertext Transfer Protocol*) com a *API REST* (*Representational State Transfer*) do *Spotify*, retornando dados como artistas, músicas e perfil do usuário. Na etapa de *front-end* será feita uma página *Web* local construída com *HTML*, *CSS* e *JavaScript*. Esta página *Web* será usada para receber do usuário a permissão para acessar os dados da sua conta na plataforma de *streaming*.

No processo para obter os dados, o criador do sistema de recomendação precisa criar uma aplicação na plataforma, que por sua vez fornecerá um *Client ID* e um *Client Secret*; por meio dessas duas informações ele conseguirá a primeira *key* chamado de *Access Token*. Essa *key* obtida pelo *flow Client Credential*<sup>4</sup> possibilita-o obter dados gerais que não envolvam dados de usuário, como dados sobre artistas, albums, músicas mais ouvidas, metadados das músicas e outros. A segunda *key*, por sua vez, é mais complicada de ser obtida, sendo necessário as duas informações anteriores e mais algumas outras, como o escopo da aplicação que determina o acesso que a aplicação pode ter dentro do perfil do usuário. Ao final desse processo, essa *key* será retornada pelo próprio *Spotify*, sendo uma autorização do tipo '*Bearer*' que será enviada no *header* de cada requisição posterior que necessite de dados do cliente. Esse processo é chamado de *Authorization code*<sup>5</sup>.

FLOW	Acessa Recursos do Usuário	Secret Key (Server Side)	Refresh Token
<i>Client credentials</i>	Não	Sim	Não
<i>Authorization code</i>	Sim	Sim	Sim

## 2.2 Trabalhos Correlatos

Diante das inúmeras dificuldades associadas à criação de sistemas de recomendação de músicas, muitos autores exploraram as formas mais eficazes para este tipo de problema.

<sup>1</sup> Documentação disponível em <https://www.r-project.org/other-docs.html>

<sup>2</sup> Documentação disponível em <https://docs.python.org/3/>

<sup>3</sup> Documentação disponível em <https://fastapi.tiangolo.com/>

<sup>4</sup> Documentação disponível em <https://developer.spotify.com/documentation/web-api/tutorials/client-credentials-flow>

<sup>5</sup> Documentação disponível em <https://developer.spotify.com/documentation/web-api/tutorials/code-flow>

Isso inclui a utilização de metadados da música, a análise das emoções do usuário e a realização de estudos para testar a eficiência e os resultados dos modelos teóricos propostos anteriormente.

O artigo “*Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*” dos pesquisadores Gediminas Adomavicius e Alexander Tuzhilin, publicado na revista do IEEE em 2005 ([ADOMAVICIUS; TUZHILIN, 2005](#)), foi um dos trabalhos pioneiros sobre sistemas de recomendação. Neste artigo, os autores propõem 3 tipos de arquitetura para conseguir resolver o problema juntamente com suas falhas inerentes. Os três tipos de arquitetura propostos foram: o modelo baseado em conteúdo, em filtragem colaborativa e o híbrido. A pesquisa busca explicar cada uma delas e determinar, para cada caso, qual a função heurística (*KNN*, *Clustering* e outros) seria melhor.

Contudo, os problemas desses modelos tornam-os limitados, não conseguindo abranger todas as necessidades do usuário. Em vista disso, após 7 anos, os pesquisadores Yading Song, Simon Dixon, and Marcus Pearce publicaram o trabalho “*A Survey of Music Recommendation Systems and Future Perspectives*”. Neste trabalho, os autores propuseram outros 3 modelos, além dos já citados, que levavam em consideração o contexto, a emoção e os metadados da música, recomendando, assim, músicas que assemelham-se estruturalmente com o mesmo volume, mesma duração e mesmo timbre, por exemplo ([SONG; DIXON; PEARCE, 2012](#)).

Muitos outros trabalhos, baseados nos artigos citados anteriormente, surgiram após 2012. Um destes trabalhos foi o de William Kraemer Aliaga, escrito em 2019. Em ‘Desenvolvimento de um sistema de recomendação musical sensível ao contexto’, Aliaga buscou criar um sistema de recomendação que a partir do contexto musical, comportamental e de ambiente fosse capaz de encontrar o gênero musical mais adequado para um determinado momento. Além disso, em sua metodologia ele implementou um pré processamento retirando dados da *API* pública do *Spotify*. A partir da obtenção dos dados, Aliaga utiliza o modelo *KNN* para conseguir recomendar músicas semelhantes ao gosto de um usuário. Nas conclusões do trabalho, o autor observou que as recomendações contextualizadas possuem maior *recall* do que as recomendações não contextualizadas ([ALIAGA et al., 2019](#)). Sua monografia distingue-se das outras por usar estratégias e abordagens interessantes, desde a forma de conseguir bases de dados de plataforma já existentes e consolidadas com uma *API* pública até o treinamento por meio do algoritmo de *KNN* e métricas de performance utilizando a distância dos cossenos.

Esta monografia é inspirada em seu trabalho tanto na extração de dados como no algoritmo usado, contudo distinguindo-se na construção do modelo (ao se considerar, por exemplo, outras abordagens estatísticas, como o PCA) e no produto final.

Outra monografia que aborda a criação de sistemas de recomendação de música

foi a dos autores Ariel Godinho e Felipe Vasconcelos, escrita em 2018 e intitulada ‘Recomendação Musical para Grupos Baseada em Modelo Híbrido’. Nesse trabalho, Godinho e Vasconcelos construíram um sistema de recomendação nomeada de *Ideal Music* por meio do algoritmo *K-Means*. A abordagem para construir o conjunto de dados foi semelhante à abordagem de Aliaga, mas utilizando um banco de dados para conseguir diminuir a taxa de requisição de *API*s e ter uma maior rapidez de resposta ao final (GODINHO; VASCONCELOS, ). Essa abordagem bem sucedida será adotada nesta monografia para não exceder a taxa limite de requisições suportadas pela plataforma de forma a guardar os dados em um banco *SQL*.

Por final, outra referência importante para esse trabalho foi o artigo ‘Rope and straw: Automatic music playlist generators’ (ALMEIDA et al., 2017) que onde busca criar um gerador de playlists, por meio do espaço originado pelo algoritmo do *t-SNE* (Figura 11) e outras técnicas os quais eles denominaram como *ROPE* e *STRAW*. A partir de duas músicas de um conjunto de dados, o algoritmo formará o menor caminho entre elas.

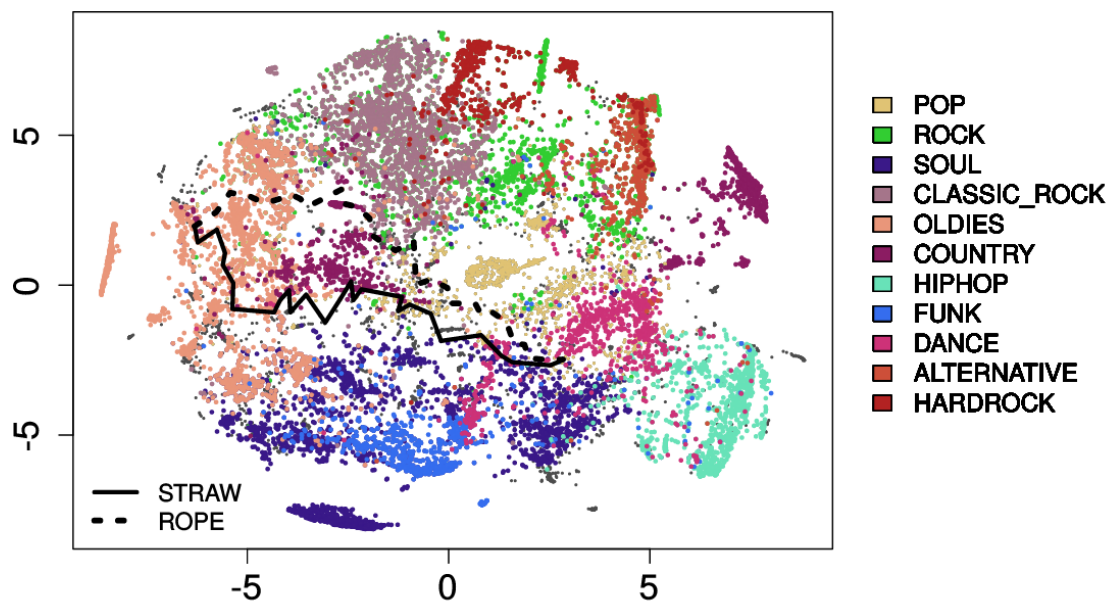


Figura 11 – *Music Playlists Generator* Fonte: Adaptado de (ALMEIDA et al., 2017)

## 3 Desenvolvimento

O trabalho foi dividido nas seguintes etapas: mineração do conjunto de dados, construção do modelo e acesso ao perfil do usuário e criação da playlist. No primeiro passo, haverá a busca de várias músicas arbitrárias por meio da *API* da plataforma de *streaming* de audio (*Spotify*) conseguindo processá-las e armazená-las em um banco de dados. Por conseguinte, haverá a construção de um modelo a partir das informações no banco de dados usando os algoritmos de aprendizado de máquina implementados em *R*. E, por fim, o usuário se conectará com a interface autorizando o seu uso. Por meio disso, o sistema obterá informações cruciais do perfil do usuário e criará a playlist em seu perfil de forma que ele possa usufruir dela.

### 3.1 Mineração do conjunto de dados

Primeiramente, o modelo precisa de dados para ser treinado. Para isso existiam duas plataformas candidatas, a *Last.fm* e o *Spotify*. Essas plataformas possuem *API*'s de fácil acesso e são bem documentadas. Em ambas existem conjuntos de dados na internet que são disponíveis para uso (como no *Kaggle*), mas a informação contida neles podem não ser confiáveis já que suas provedoras podem ter atualizado elas.

Por conta disso, construir a base de informação do zero com os elementos da *API* foi a solução e a plataforma escolhida foi a do *Spotify*. Para começar, foram selecionadas duas etapas de consumo dessa *API*: uma etapa para extrair dados genéricos como artistas, composição, metadados de cada música, ou seja, recursos que não são ligados com um usuário; outra etapa para extrair os dados do cliente como seus artistas favoritos, músicas curtidas e outras informações. Essa última etapa será mais explicada na seção 3.3.

Na primeira fase, a *API* foi consumida através de um código na linguagem *Python* seguindo os endpoints dados pela documentação da plataforma. Nessa parte, a monografia inspirou-se no trabalho de Aliaga et al. (2019). Para conseguir realizar as requisições nos *endpoints* presentes na documentação da plataforma é primeiro necessário criar uma aplicação na plataforma de *streaming*, o qual fornecerá um *Client ID* e um *Client Secret* como mostrado na Figura 12.

Por meio da documentação do *Spotify*<sup>1</sup> temos acesso a url base '*api.spotify.com/v1/*' a qual confere permissão a uma grande quantidade de dados. Para isso houve a filtragem dos *endpoints* que serão usados, parte na mineração e parte na geração da playlist com

<sup>1</sup> Documentação disponível em <https://developer.spotify.com/documentation/web-api>



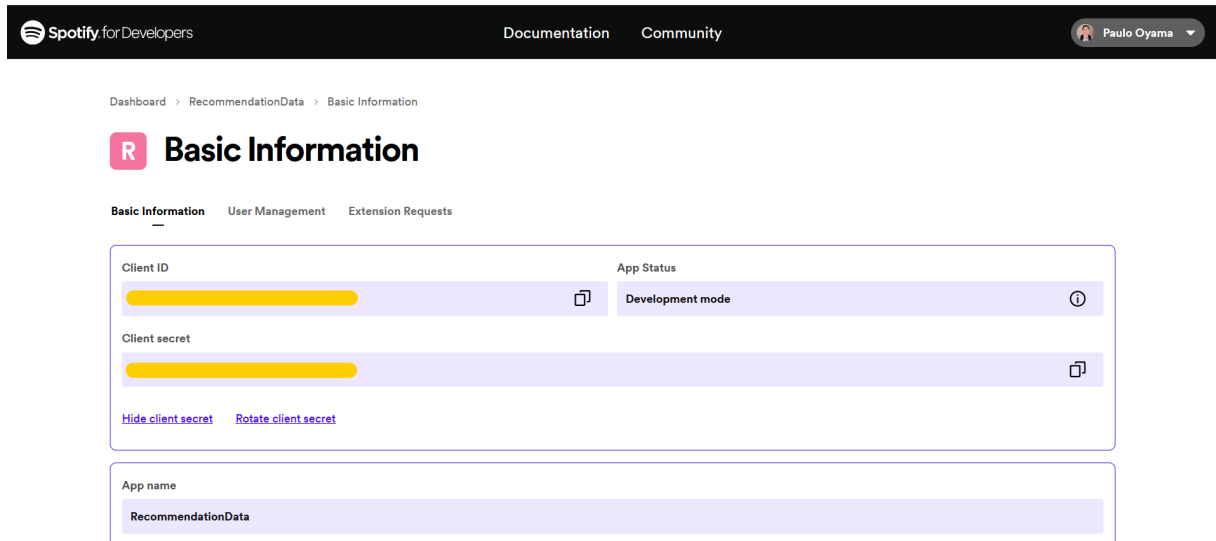


Figura 12 – Exemplo de aplicação no site do *Spotify*. Fonte: Do autor

auxílio do pacote *Spotipy*<sup>2</sup> versão 2.22.1.

Nome	Endpoints	Tipo
<i>Search Items</i>	search	GET
<i>Related Artists</i>	artists/id/related-artists	GET
<i>Artists Top Tracks</i>	artists/id/top-tracks	GET
<i>Track Audio Features</i>	audio-features/id	GET
<i>User Profile</i>	me	GET
<i>User Top Tracks</i>	me/top/type	GET
<i>Create Playlist</i>	users/{user_id}/playlists	POST
<i>Add items to Playlist</i>	playlists/{playlist_id}/tracks	POST

Por conta da necessidade de um conjunto extenso e diverso de dados, a parte de mineração<sup>3</sup> dividiu-se em achar artistas de diferentes países, gêneros, ritmos e tonalidades, depois conseguir os artistas relacionados a esses artistas por meio da *API*, então para cada artista conseguir suas melhores músicas e também seus metadados. Além disso, com essa abordagem, pode-se contornar o problema chamado de *Cold Start* descrito em (ADOMAVICIUS; TUZHILIN, 2005), o qual explica que no início de um sistema de recomendação, devido à escassez de informações na base, as recomendações geradas tendem a ser imprecisas, melhorando à medida que mais dados são inseridos no banco.

A partir de agora, tomaremos como referência a Figura 13 para descrever os passos referentes à mineração dos dados. A primeira parte demonstrada nesta figura é feita pelas setas 1, 2 e 3. A seta 1 é uma lista feita manualmente através de pesquisa em sites por

<sup>2</sup> Documentação disponível em <https://spotipy.readthedocs.io/en/2.22.1/>

<sup>3</sup> Disponível em [https://github.com/PauloOyama/Undergraduate-Final-Project/tree/main/Scrapping\\_api](https://github.com/PauloOyama/Undergraduate-Final-Project/tree/main/Scrapping_api)



artistas que fazem sucesso, por exemplo pesquisas como *Top Hits de 2023* ou *Cantores de Jazz*; a seta **2** é a requisição para *API* para buscar esses cantores e então guardá-los em um *csv* como mostrado em **3**.

A segunda parte serve para enriquecer o arquivo de dados anteriormente criado, de forma a aumentá-lo rapidamente. Assim, artistas relacionados aos que já estão nele também serão inseridos sendo que a descoberta desses novos artistas é feita pela *API* da plataforma. Ao final, estes novos dados serão adicionados ao conjunto. Esse processo é mostrado pelas etapas **4**, **5** e **6**.

A terceira parte tem como objetivo minerar as faixas mais ouvidas de cada artista presente no arquivo ‘Artistas’, ou seja, para cada artista presente no arquivo haverá uma chamada de *API* e seu retorno será guardado em um arquivo ‘Músicas Mais Ouvidas’, sendo mostrado por **7**, **8** e **9**.

A parte final será usar cada música presente no passo **9** para obter seus metadados, os quais serão usados para a etapa descrita na seção 3.2 (passos **9**, **10** e **11**).

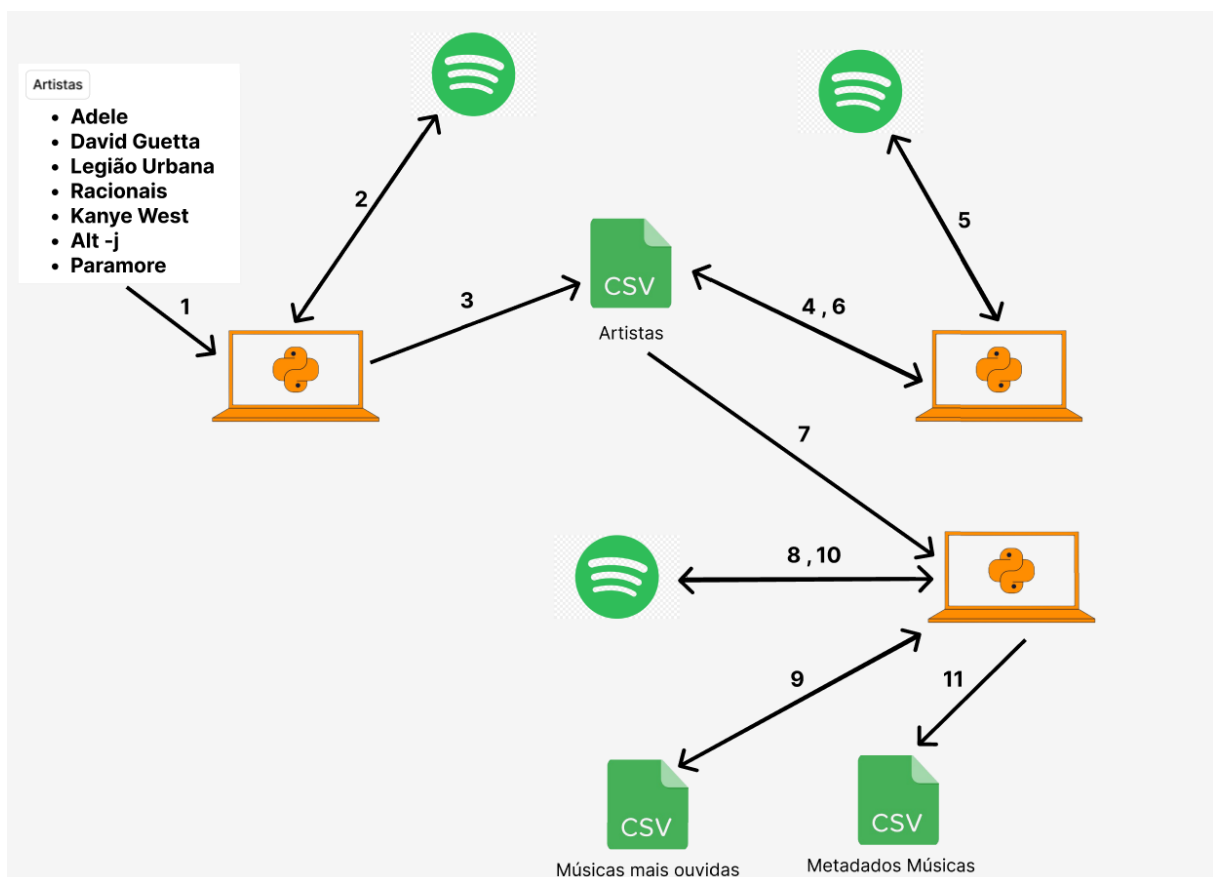


Figura 13 – Flow da mineração com a *API* do *Spotify*. Fonte: Do autor

O uso de dados por meio de arquivos faz-se satisfatório para pesquisas e estudos locais, contudo o intuito desse trabalho é também ser escalável para possíveis melhorias. Por conta disso, usaremos um banco de dados *MySQL* na plataforma *Amazon Web Service*

(AWS) usando o serviço para gerenciamento de banco de dados chamado de *RDS*<sup>4</sup>.

O banco criado terá o modelo de entidade relacionamento seguindo os padrões mostrados na Figura 14, onde a *Table Artists* refere-se ao arquivo ‘Artista’ (Figura 13), a *Table TopTracks* refere-se ao arquivo ‘Músicas mais ouvidas’ e *Table TrackFeatures* refere-se ao arquivo ‘Metadados Músicas’.

Esse banco poderá ser usado para reduzir a taxa de requisições entre o programa e a *API* do *Spotify*, a qual contém uma taxa limite de requisições por tempo (passada essa quantidade, a aplicação levará uma penalização de 30 *seg* de *timeout*). Essa camada foi inspirada no trabalho do (GODINHO; VASCONCELOS, ) já que, por meio dela, é possível um melhor gerenciamento de instâncias e rapidez de resposta por conta do paralelismo.

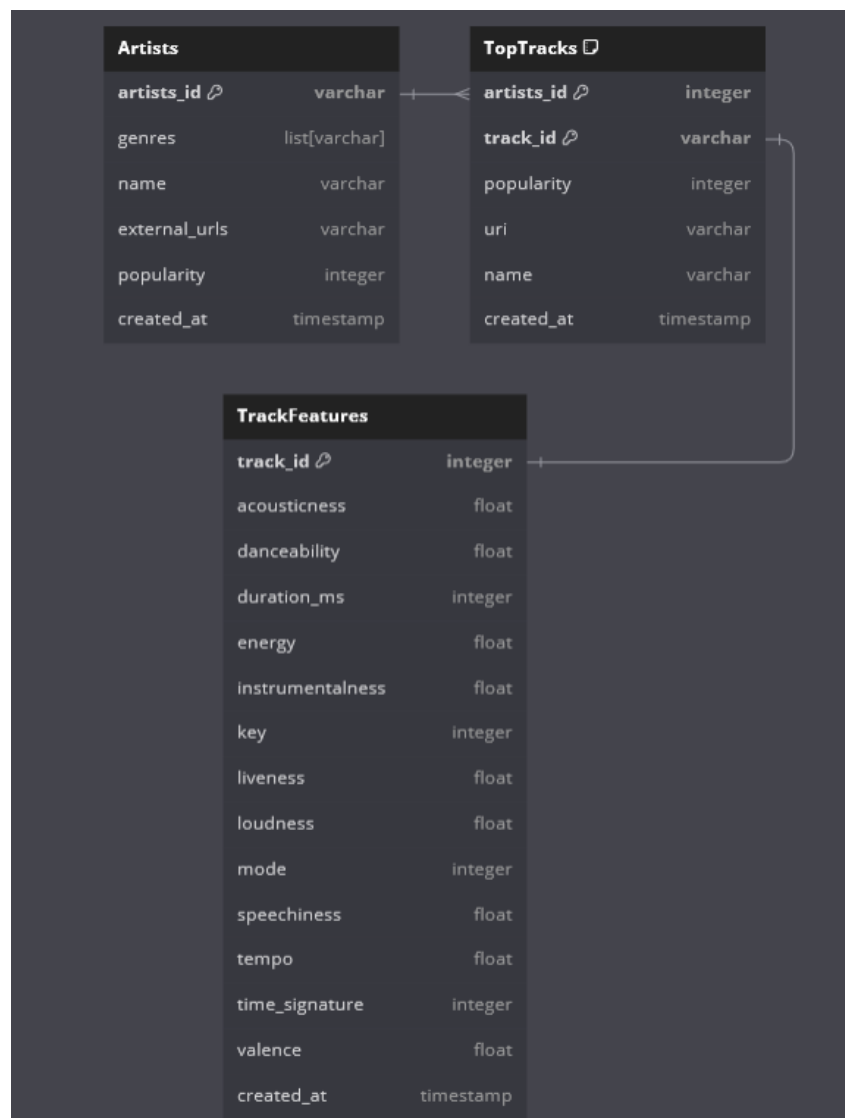


Figura 14 – Estrutura relacional dos dados. Fonte: Do autor

<sup>4</sup> Disponível em <https://aws.amazon.com/pt/rds/>

## 3.2 Treinamento com os dados

Para o treinamento com os dados houve a necessidade de inclusão de novas variáveis no arquivo. A popularidade de uma música, por exemplo, vem de *Table TopTracks*. Observa-se sua utilidade no contexto do gosto do cliente, já que um usuário pode escutar músicas com popularidades maior e portanto o algoritmo dará uma playlist com músicas populares enquanto que outra pessoa pode ouvir estilos mais desconhecidos e portanto o algoritmo deve levar isso em conta, criando uma playlist com músicas menos populares.

O processo para construção do sistema de recomendação é feito por meio de manipulação e visualização dos dados e refinamento; então, o algoritmo de aprendizado de máquina será utilizado para gerar o produto, como mostrado na Figura 15. Essas etapas serão discutidas abaixo e parte delas serão feitas em *R* com o pacote *ggplot2*<sup>5</sup>. Este pacote contém funções que permitem a construção de gráficos extremamente bonitos e informativos.

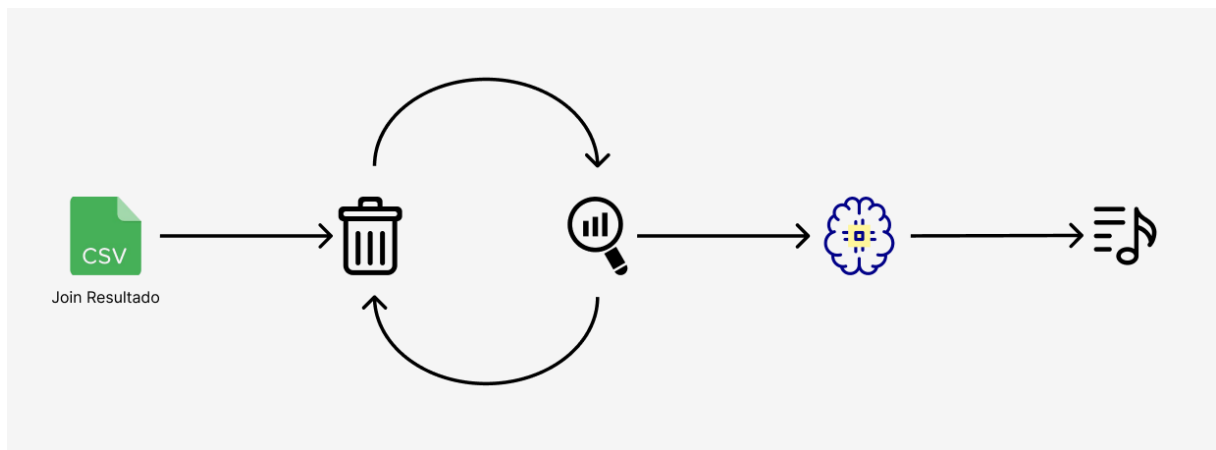


Figura 15 – Flow do treinamento com os dados. Fonte: Do autor

Primeiramente, o arquivo *Join Resultado* obtida pelo *inner join* entre as *Tables TrackFeatures* e *TopTracks* pela *key track\_id* apresenta-se como um conjunto de dados da seguinte estrutura.

<sup>5</sup> Disponível em <https://ggplot2.tidyverse.org/reference/>

Nome	Tipo	Descrição
<i>Name</i>	chr	string
<i>Track Id</i>	chr	string
<i>URI</i>	chr	string
<i>Mode</i>	int	range [0,1]
<i>Key</i>	int	range [-1,...,11]
<i>Time signature</i>	int	range [3,...,7]
<i>Popularity</i>	int	range [0...100]
<i>Acousticness</i>	float	range [0, ... ,1.0]
<i>Danceability</i>	float	range [0 ... 1.0]
<i>Energy</i>	float	range [0 ... 1.0]
<i>Instrumentalness</i>	float	range [0 ... 1.0]
<i>Loudness</i>	float	range [-60.0 ... 0]
<i>Liveness</i>	float	range [0 ... 1.0]
<i>Speechiness</i>	float	range [0 ... 1.0]
<i>Valence</i>	float	range [0 ... 1.0]
<i>Duration ms</i>	float	range [0 ...Infinity]
<i>Tempo</i>	float	range [0 ... Infinity]
<i>Created at</i>	datetime	%YY-%MM-%DD hh:mm:ss

Para o treinamento, o uso de valores de string não serão necessários, portanto variáveis do tipo 'chr' podem ser excluídas visto que são links *https*, nome da música e *endpoints* para a plataforma. Além disso, quase todas as variáveis inteiras podem ser descartadas por não ter um percentual alto na representatividade dos dados, somente deixando o atributo de popularidade que pode melhorar o produto criado. E por final, o campo de data também pode ser excluído servindo somente como guia nas tabelas do banco de dados. Retirando todas essas colunas, haverá um conjunto de 11 atributos para serem analisados com o objetivo de gerar um modelo eficiente de modo a retornar músicas com características semelhantes.

Uma das dificuldades encontradas já de início foi não haver uma variável resposta visto que, para uma música pode-se encontrar diversas outras músicas que sejam semelhantes e que sejam uma resposta aceitável. Portanto, para uma análise exploratória inicial foi usado a matriz de correlação para encontrar variáveis correspondentes e tentar entender o comportamento dos dados por meio dos gráficos como mostrado na Figura 16.

Essa figura levantou hipóteses de que gráficos em duas dimensões construídos a partir de combinações dois a dois dos atributos do conjuntp não seriam suficientemente bons para entendermos os dados. Por conta disso, houve a necessidade de alguma técnica que pudesse levar em conta  $N$  atributos,  $N > 2$ , que seriam mapeados em 2 dimensões sem a perda de informação.

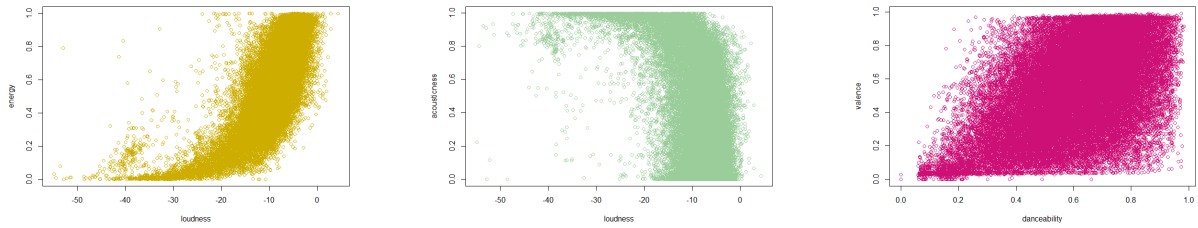


Figura 16 – Comportamento dos dados por duas dimensões independentes. Fonte: Do Autor

Para tanto, baseado no artigo de Almeida et al. (2017) o qual utiliza-se do *t-SNE* para visualização e geração de um espaço  $R^2$ , o conjunto de treinamento também utilizou do mesmo método para observação do comportamento dos 11 atributos por meio da biblioteca *Rtsne*<sup>6</sup>.

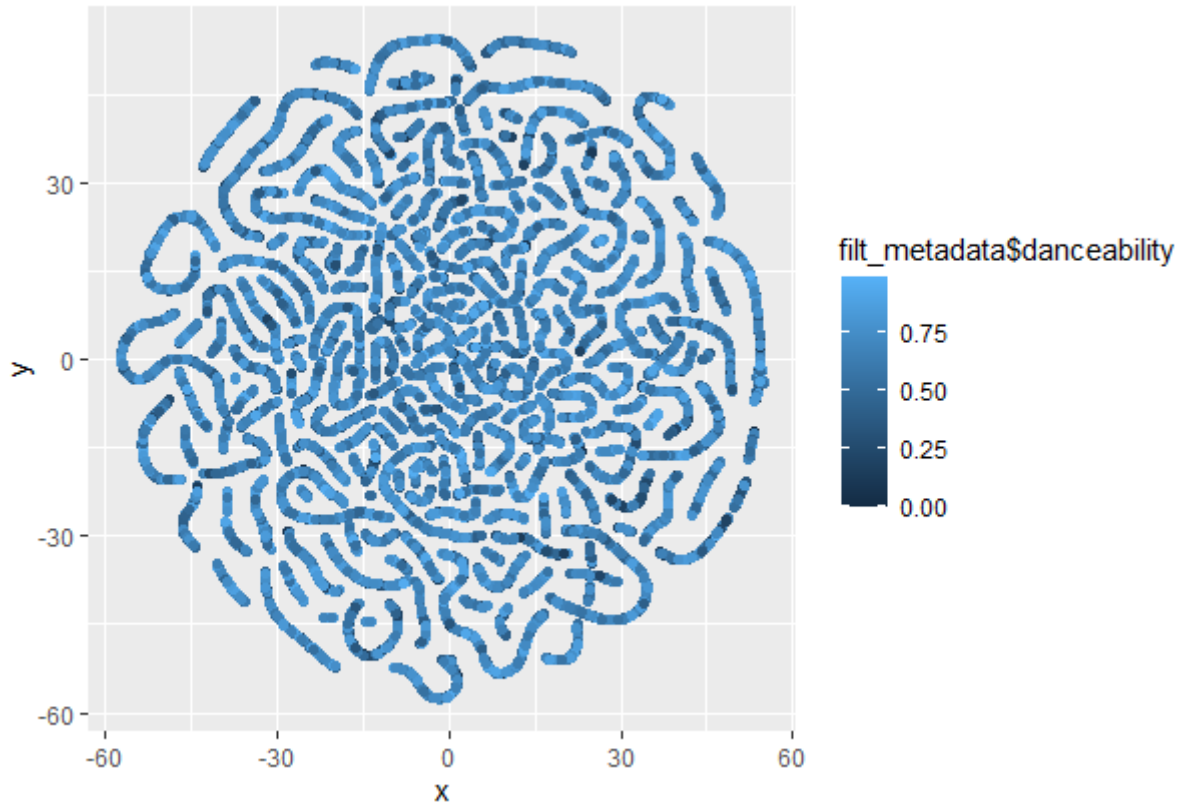


Figura 17 – *t-SNE* plot de  $R^{11}$  em  $R^2$ . Fonte: Do autor

A Figura 17 mostra o comportamento inicial de aglomerados por formarem ilhas, contudo é ainda superficial para uma segmentação clara.

Uma hipótese, foi que os artistas escolhidos na fase 1 da Figura 13 não foram balanceados, por exemplo, haviam mais cantores rappers do que sertanejos ou mais cantores de funk do que mpb. Por meio dessa hipótese, o estudo tentou trazer os gêneros(*genres*)

<sup>6</sup> Disponível em <https://cran.r-project.org/web/packages/Rtsne/index.html>

presentes na *Table Artists*. Contudo, ao se tentar levar em consideração o gênero musical de cada artista, um obstáculo foi encontrado. Cada artista está classificado em vários gêneros. Por conta disso, enquadrar os artistas em gêneros é um desafio a ser estudado e uma ideia que será considerada em trabalho futuro.

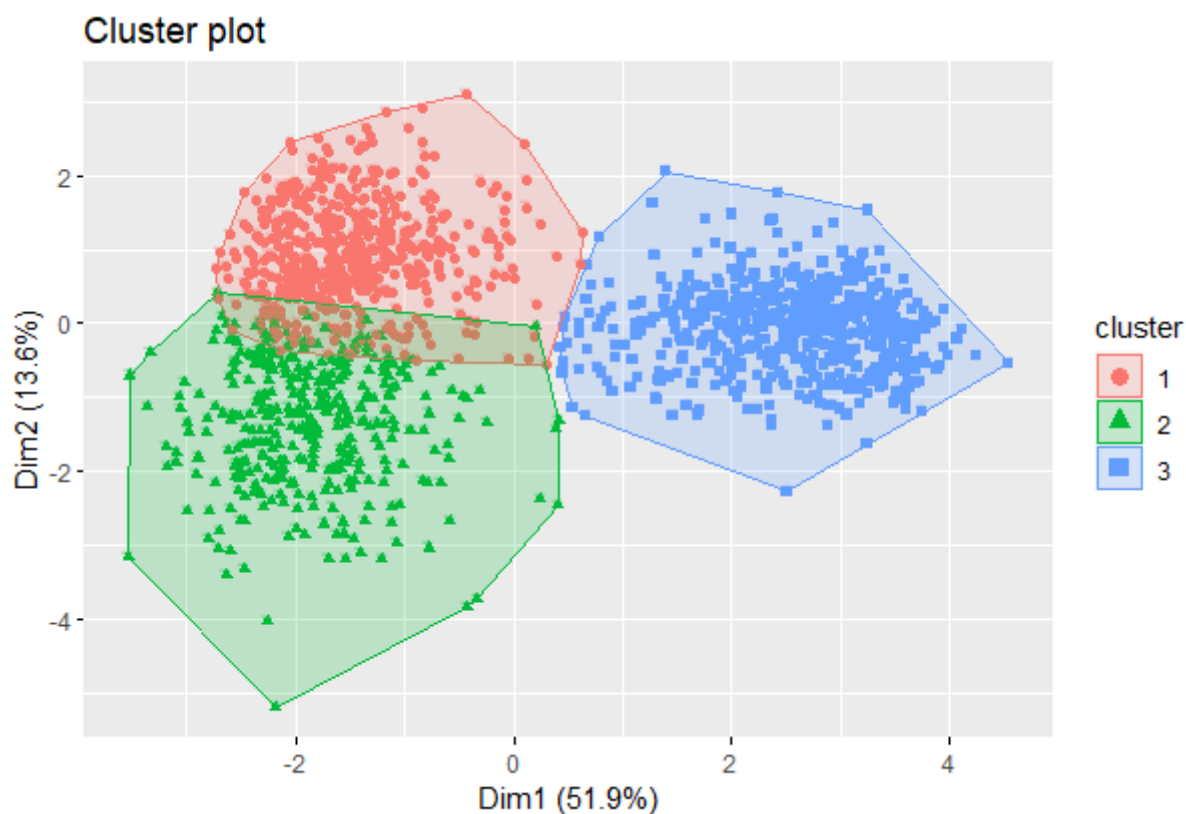


Figura 18 – Clusterização com 3 gêneros. Fonte: Do autor

Um estudo levando em consideração somente 3 gêneros musicais (*hip hop*, *classical* e *metal*) foi realizado. Neste estudo, as variáveis numéricas destas músicas foram utilizadas para a construção de um modelo não supervisionado, o *KMeans*, com o objetivo de agrupar as músicas semelhantes. A Figura 18 mostra o resultado deste processo. Verificou-se que cada um dos aglomerados compreendia exatamente, em sua maioria, observações de apenas um dos tipos musicais considerados, evidenciando assim, que estes três gêneros possuem características que tornam duas músicas de um mesmo tipo próximas e músicas de tipos diferentes distantes. Entretanto, ao se tentar aplicar este mesmo processo mais gêneros musicais, a separação em aglomerados já não ficou razoavelmente bem dividida, muito provavelmente por conta dos problemas de classificação dos gêneros apontados anteriormente. Por conta disso, foi necessário achar outros caminhos que poderiam ser tomados.

Em seguida, tentou-se reduzir a dimensão do conjunto com o objetivo de se ganhar em eficiência e também em facilitar as análises estatísticas. Para esta etapa, utilizou-se do algoritmo de *PCA*. O *PCA* (Principal Component Analysis) é uma técnica de redução de

dimensionalidade usada para identificar padrões nos dados e expressá-los em termos de suas variáveis principais, denominadas como componentes principais. O objetivo do *PCA* é simplificar a complexidade dos dados, mantendo o máximo de informação possível. O resultado da aplicação do *PCA* a partir dos pacotes *factoextra*<sup>7</sup> e *FactoMineR*<sup>8</sup> pode ser visto pela Figura 19.

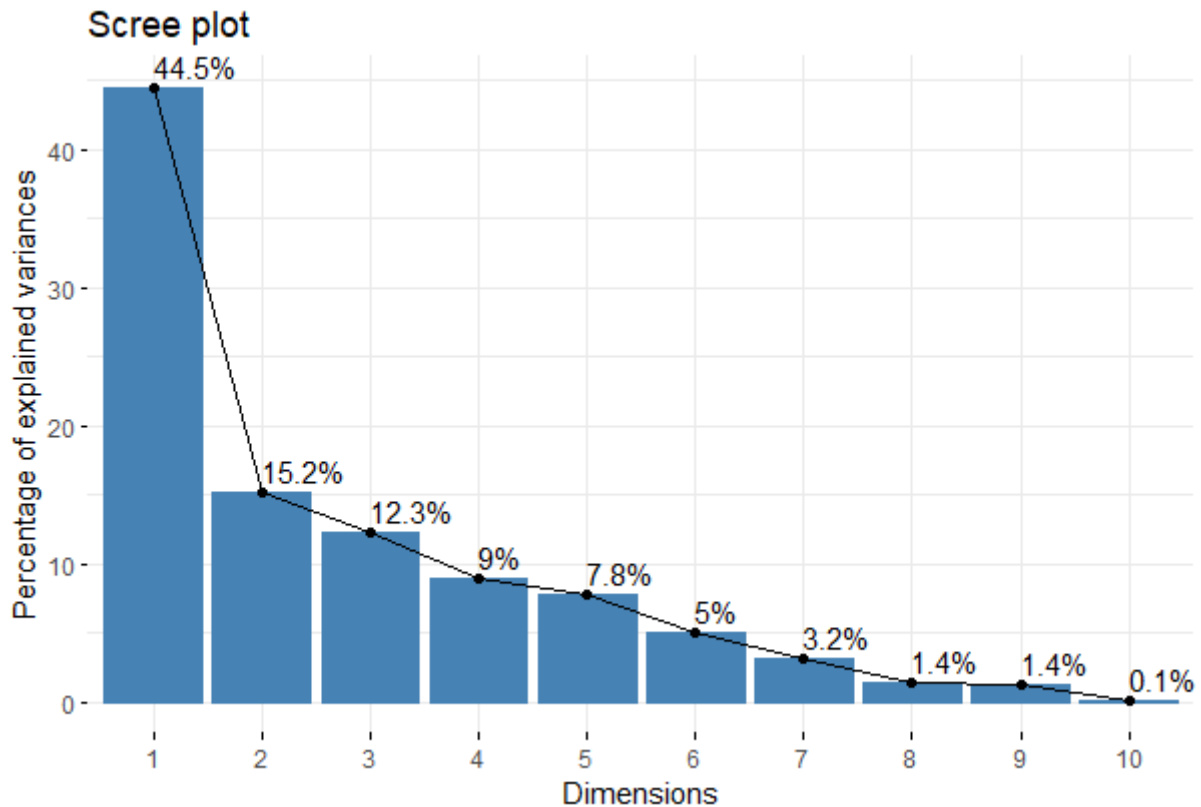


Figura 19 – Variância por número de dimensões no sistema. Fonte: Do autor

Essa figura mostra que apenas uma componente principal consegue explicar 44,5% do conjunto original enquanto que a segunda componente principal explica 15,2%, isto é, juntas estas duas componentes explicam 59,7%. Para ver a variância das 11 dimensões e como elas se comportam, o algoritmo fornece a proporção cumulativa (*Cumulative Proportion*) por componentes principais.

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5	Comp.6
Standard deviation	2.213	1.295	1.164	0.993	0.928	0.744
Proportion of Variance	0.445	0.152	0.123	0.089	0.078	0.050
Cumulative Proportion	0.445	0.597	0.720	0.810	0.889	0.939

<sup>7</sup> Disponível em <https://cran.r-project.org/web/packages/factoextra/index.html>

<sup>8</sup> Disponível em <https://cran.r-project.org/web/packages/FactoMineR/index.html>

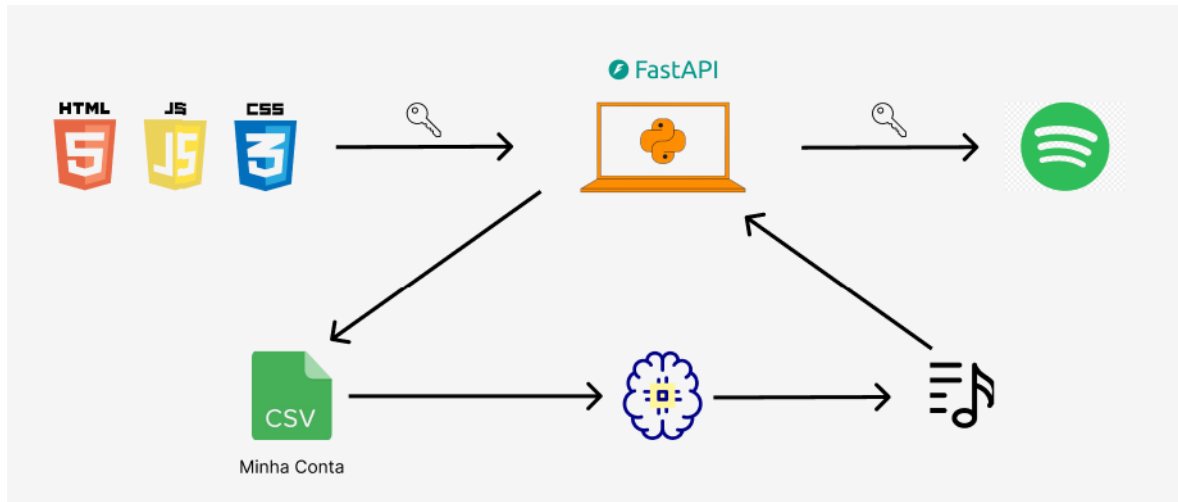


Figura 20 – Fluxo do contato do cliente. Fonte: Do autor

	Comp.7	Comp.8	Comp.9	Comp.10	Comp.11
Standard deviation	0.590	0.396	0.385	0.108	1.440740e-08
Proportion of Variance	0.031	0.014	0.013	0.001	1.887028e-17
Cumulative Proportion	0.971	0.985	0.998	1.000	1.000000e+00

Logo, observa-se que reduzir a dimensionalidade de 11 para 5 ( $\mathbb{R}^{11} \Rightarrow \mathbb{R}^5$ ) consegue retratar bem o conjunto original pois com somente 5 atributos pode-se explicar  $\pm 89\%$  das informações. Portanto, o conjunto final terá 5 atributos somente, sendo que cada um destes atributos é uma combinação linear dos 11 atributos das informações originais.

Após essa etapa, os dados estarão prontos para serem utilizados em uma busca pelos vizinhos mais próximos. Inicialmente, tentamos implementar o algoritmo KNN (K-Nearest Neighbors) utilizando o pacote *class*<sup>9</sup>. No entanto, constatamos que sua implementação não é escalável, pois leva vários minutos para calcular as distâncias entre 40.000 músicas. Além disso, é instável a mudanças, como inserção e remoção de dados, tornando-o inviável para uso em cenários reais com clientes, onde o tempo de resposta é crucial. Portanto, a alternativa foi implementar o algoritmo do *KD Tree* pelo pacote *less*<sup>10</sup> já que com sua estrutura, ele provê uma construção rápida e tem capacidade de adaptação (BHATIA et al., 2010) dando ao usuário uma resposta em tempo aceitável e uma escalabilidade maior na parte arquitetural.

### 3.3 Acesso ao perfil e criação da playlist

Após realizar a construção do modelo, é necessário os dados do usuário para serem usados como entrada na árvore e para depois se fazer o upload da playlist na conta do

<sup>9</sup> Disponível em <https://cran.r-project.org/web/packages/class/class.pdf>

<sup>10</sup> Disponível em <https://search.r-project.org/CRAN/refmans/less/html/KDTree.html>



*Spotify* da pessoa. Para isso, como mostrado na Figura 20, é preciso primeiro a autorização do usuário em algum endereço de IP, seja ele *localhost* ou hospedado em domínios públicos. Nessa monografia foi construída uma página simples como indicado na figura com *HTML*, *CSS* e *JavaScript* e feito localmente.

Por meio dele serão feitos dois processos, o de autorizar o acesso aos dados do usuário na plataforma e também de criar a playlist na conta dele. Para o primeiro é necessário clicar no botão ‘*AUTORIZAR APLICAÇÃO*’ mostrado na Figura 21 e depois haverá um redirecionamento para a página do *Spotify* onde lá o usuário terá que aceitar os termos de uso da aplicação. Essa parte é feita pela própria plataforma de *streaming*.

Após esse processo, o *Spotify* redirecionará o usuário para uma url definida pelo autor da aplicação, junto com o *Authorization Code*, que será enviado no cabeçalho das requisições para obter informações do usuário, criar uma playlist e adicionar músicas a ela. Essa chave será utilizada pelo framework *FastAPI* para obter as músicas mais ouvidas pelo usuário, por meio de um servidor local executado no endereço `http://127.0.0.1` e na porta 8000. Essa chamada à API acontecerá automaticamente após a conclusão do redirecionamento, gerando um arquivo *csv* com os dados do cliente. Esses dados, por sua vez, seguem a mesma estrutura da tabela 3.2, e serão usados como entrada na árvore *KD Tree* com  $K = 2$  para gerar uma lista de músicas que é a saída do algoritmo.

Esses dados por sua vez seguem a mesma estrutura da tabela 3.2, e serão usados como entrada na árvore *KD Tree* com  $K = 2$  para gerar uma lista de músicas que é a saída do algoritmo.

Após a geração desse produto, um código em *Python* utilizando *FastAPI* realizará duas requisições POST para o perfil do usuário: uma para criar uma playlist e outra para adicionar cada música à lista criada, utilizando o ID da música (Track Id). Esse processo ocorre quando o usuário clica em ‘*CREATE PLAYLIST*’, de acordo com a ilustração da Figura 21.



Figura 21 – Página *Web* local. Fonte: Do autor

## 4 Resultados

O resultado gerado é um sistema híbrido que considera tanto os metadados da música quanto o perfil do usuário para recomendações musicais, aumentando, assim, as chances de descoberta de novas músicas que estejam em sintonia com seu gosto pessoal. A partir do link <https://github.com/PauloOyama/Undergraduate-Final-Project> pode-se acessar o código da implementação do sistema. Além disso, o sistema gera um produto real, permitindo que os usuários o utilizem no dia a dia para descobrir e ouvir músicas recomendadas, conforme ilustrado na Figura 22. A arquitetura do sistema é escalável, o que possibilita futuras pesquisas para agregar mais valor e expandir ainda mais o produto.

Por fim, outro resultado importante é que, por meio de testes manuais (ouvir as

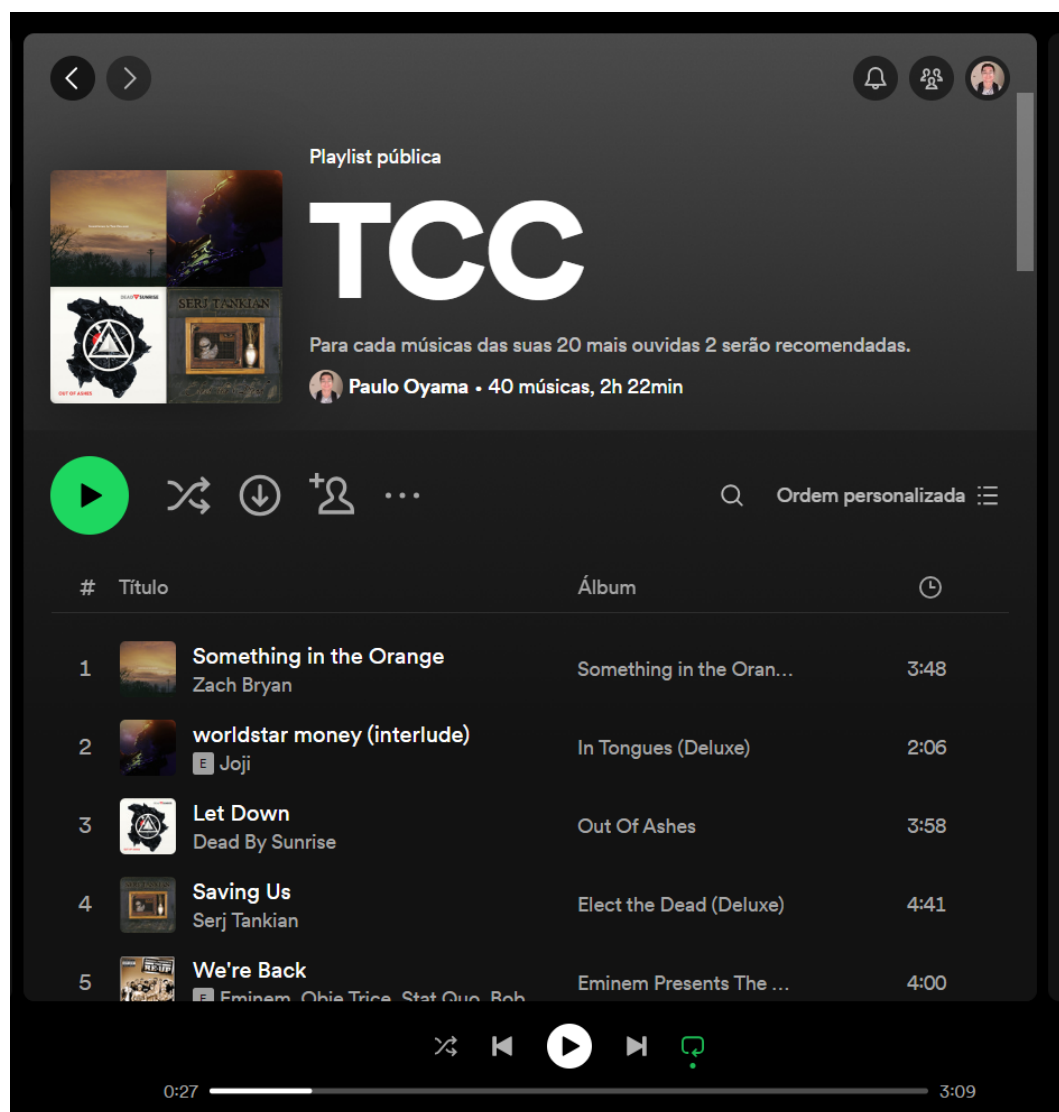


Figura 22 – Prduto gerado na conta do usuário. Fonte: Do autor

músicas recomendadas e compará-las com as músicas de entrada), observou-se que o sistema começou a identificar conceitos básicos do som, conforme descrito em (MILETTO et al., 2004). Por exemplo, para músicas que contêm apenas piano, como as de Beethoven ou Mozart, o algoritmo também recomendará músicas que apresentam apenas o instrumento piano. Da mesma forma, para músicas de gêneros bem definidos, como rock, o algoritmo também recomendará músicas desse mesmo gênero. Além disso, em muitos casos, as músicas recomendadas estarão no mesmo idioma da música de entrada, o que corrobora a eficiência do modelo e das escolhas feitas na sua construção.

## 5 Conclusão

Nesse trabalho foi apresentado o conceito de sistemas de recomendação, modelos híbridos, algoritmos de aprendizado e ferramentas estatísticas e matemáticas para a limpeza e filtragem dos dados, vindo desde a necessidade vista para criação desse sistema, passando por sua construção e indo até o produto gerado.

Portanto, essa monografia teve como objetivo mostrar que é possível a construção de um sistema híbrido de recomendação com os dados da plataforma do *Spotify*, e juntamente com a construção do sistema mostrar o passo a passo matemático e estatístico por trás de cada escolha, tornando o replicável para qualquer um, por meio de técnicas como *PCA*, *KD-Tree* e *t-SNE* assim como ferramentas estruturais como *Python*, *R*, *HTML*, *CSS*, *JS* ( *front-end* ) e *FastAPI* ( *back-end* ).

Ainda sim, este trabalho trouxe diversas partes dos trabalhos correlatos e *surveys*, de modo a juntar a teoria dos *surveys* sobre modelos de recomendação juntamente com a implementação de um sistema com a geração de um produto palpável. Além disso, buscou-se implementar dos artigos correlatos as partes que são mais relevantes ao objetivo do trabalho tornando o sistema eficiente, escalável e viável para qualquer público por meio dos autores inspirados.

E também, deixando por meio desta monografia uma adição a literatura de sistemas de recomendação no que consta ao pensamento matemático sobre as escolhas das alternativas tomadas por meio de gráficos, algoritmos de redução de dimensionalidade e técnicas estatísticas o qual não possui nas referências, além disso explicitando a construção e o caminho tomado, os 'por quê' e as respostas, de modo a abrir um caminho que artigos futuros podem basear-se para seguir ou inspirar.

Além disso, deve-se salientar as limitações do presente trabalho sendo eles,

1. Input manual dos artistas;
2. Armazenamento local por meio de arquivos *csv*;
3. Números de *testers* para a aplicação é limitado pelo *Spotify*;
4. Desbalanceamento de gêneros;
5. Recomendações improváveis em gêneros não definidos.

O input manual dos artistas apresenta uma limitação significativa, pois é um processo manual e não automático, podendo portanto demorar muito tempo e ocorrer um

desbalaceamento de gênero, pendendo para os gostos daquele que estará coletando os dados. Além disso, o armazenamento local por *csv* é eficiente para estudos pequenos mas pouco escaláveis para grandes quantidades de dados e para tanto uma proposta de melhoria seria o uso de arquivos com orientação colunar como o *parquet*.

Além disso, o número de testadores para a aplicação é limitado devido à plataforma de desenvolvimento do Spotify, que não é destinada à publicação. Essa restrição também afeta a quantidade de requisições que podem ser feitas em um determinado intervalo de tempo. Ao final, as últimas duas são mais sutis de serem observadas, mas podem ser vistas no produto gerado. A primeira foi observada no momento das construções dos gráficos como discutido anteriormente e depois também nas músicas da playlist que mostram uma tendência para músicas do gênero funk ou rap mesmo não sendo o gênero de entrada. A segunda é observada nas músicas recomendadas para conteúdos com gêneros turvos que tendem a ter características de mais de um gênero, por exemplo o *indie*. Para pessoas que ouvem este tipo de música, o algoritmo pode recomendar dois gêneros muito distintos que não correspondentes ao gosto do usuário.

Por conclusão, a monografia também possui espaço para trabalhos futuros. Dentre eles, podemos citar:

1. Adicionar mais uma característica no modelo híbrido:

A *API* disponibilizada pelo *Spotify* é vasta ao ponto de conseguir obter os amigos do usuário e suas playlists por meio de *endpoints* diferentes das citadas na tabela 3.1. Portanto uma vertente possível para essa monografia seria um sistema híbrido que agregaria os *metadados*, informações do perfil do usuário e o perfil do seus amigos. Este sistema é chamado na literatura de *collaborative filtering* (SONG; DIXON; PEARCE, 2012).

2. Possibilitar mais de um algoritmo de aprendizado de máquina:

A ideia seria usar um segundo algoritmo de aprendizado de máquina supervisionado ou não, como meio de mesclagem entre as duas, resultando em um produto com características de dois algoritmos diferentes que podem se complementar como feito no desenvolvimento do trabalho de (GODINHO; VASCONCELOS, ).

3. Usar os valores gerados pelo *t-SNE*;

Um outro tipo de abordagem seria trocar o *PCA* pelo uso dos valores gerados pelo *t-SNE*, de forma a ter um conjunto de dados de dimensões  $\mathbb{R}^2$  e podendo ser implementado juntamente com um *KNN* novamente ou outras técnicas mais apropriadas.

4. Implementar parâmetros de medida do algoritmo:

Gerar parâmetros de validação como acurácia, *f-score* ou *recall*; isso poderia ser feito através da criação de um conjunto de teste dentro do de treino, por exemplo. Sabendo que uma música *A* tem como vizinhos mais próximos *B* e *C*, todos pertencentes ao conjunto de treino, retira-se *A*, colocando-a em um conjunto de teste; espera-se, então que um bom sistema seja capaz de recomendar as músicas *B* e *C* para o usuário.

5. Encontrar o melhor *K*;

A partir da ideia anterior, também é possível achar o melhor *K* para o determinado conjunto, utilizando as métricas acima e também *cross-validation* sobre o conjunto de treino.

6. Estruturar a arquitetura no *cloud*;

Estruturar uma arquitetura *cloud* tornando o produto mais escalável e disponibilizando-o online para usuários fora da rede *localhost*. Uma proposta de arquitetura usando a plataforma *AWS* é mostrada na Figura 23. Nela a camada de scrapping pode ser feita pelo serviço *Lambda* juntamente com o *API Gateway*. A partir de armazenamento com *RDS* e *MySQL*, o algoritmo de aprendizado de máquina pode ser hospedado pelo *SageMaker*, integrando a saída e entrada com o serviço de *Lambda* e uma camada de monitoramento e *logs* para caso seja preciso. neste caso, a ferramenta *FastAPI* será substituída pelo *API Gateway* e a parte de *front-end* pode ser armazenada e disponibilizada com *IP* público através do *S3*.

7. Retreinar o modelo com novos valores:

Com a arquitetura anterior, é possível realizar o retreinamento do modelo, de forma aumentar a assertividade da resposta e também a inserção de novas músicas alimentando a base de dados. Isso resulta em um aumento das possibilidades de saída para uma determinada música. O retreinamento não precisa ocorrer toda vez que há uma inserção; ele pode ser feito em intervalos de tempo definidos, aproveitando a estrutura de árvore na qual os dados estão inseridos.

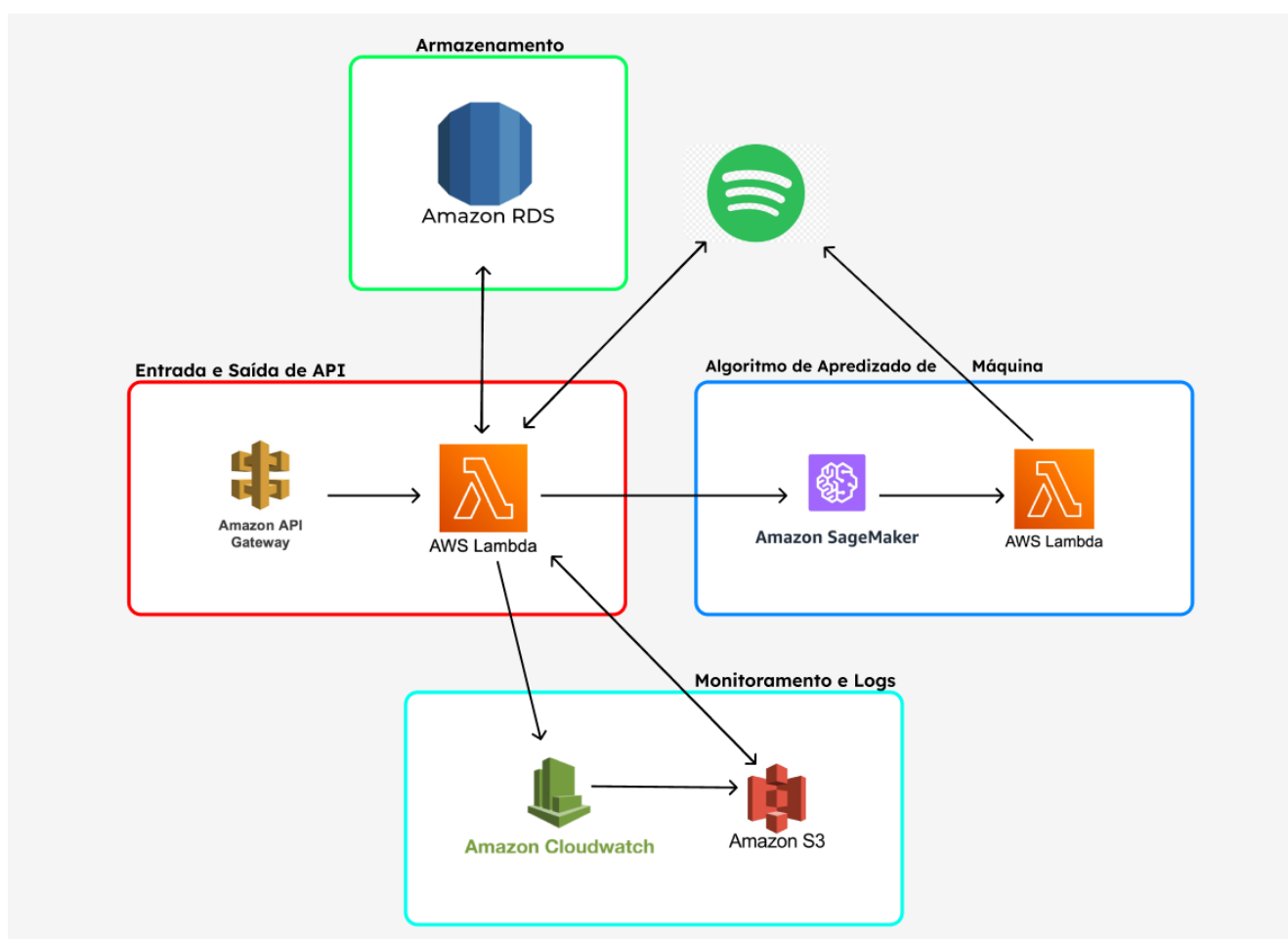


Figura 23 – Possível arquitetura para implementação futura. Fonte: Do autor



# Referências

- ADOMAVICIUS, G.; TUZHILIN, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. **IEEE transactions on knowledge and data engineering**, IEEE, v. 17, n. 6, p. 734–749, 2005. Citado 3 vezes nas páginas 14, 28 e 31.
- ALIAGA, W. K. et al. Desenvolvimento de um sistema de recomendação musical sensível ao contexto. Florianópolis, SC., 2019. Citado 3 vezes nas páginas 9, 28 e 30.
- ALMEIDA, M. A. de; VIEIRA, C. C.; MELO, P. O. V. de; ASSUNÇÃO, R. M. Rope and straw: Automatic music playlist generators. In: SBC. **Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web**. [S.l.], 2017. p. 204–208. Citado 4 vezes nas páginas 6, 26, 29 e 36.
- BELLMAN, R.; CORPORATION, R.; COLLECTION, K. M. R. **Dynamic Programming**. Princeton University Press, 1957. (Rand Corporation research study). ISBN 9780691079516. Disponível em: <<https://books.google.com.br/books?id=wdtoPwAACAAJ>>. Citado na página 24.
- BHATIA, N. et al. Survey of nearest neighbor techniques. **arXiv preprint arXiv:1007.0085**, 2010. Citado 3 vezes nas páginas 18, 19 e 39.
- FRADKOV, A. L. Early history of machine learning. **IFAC-PapersOnLine**, Elsevier, v. 53, n. 2, p. 1385–1390, 2020. Citado na página 15.
- GODINHO, A.; VASCONCELOS, F. Recomendação musical para grupos baseada em modelo híbrido. Citado 4 vezes nas páginas 9, 29, 33 e 45.
- JAMES DANIELA WITTEN, T. H. G.; TIBSHIRANI, R. **An Introduction to Statistical Learning with Applications in R**. 1. ed. [S.l.]: Springer, 2013. Citado na página 16.
- KO, H.; LEE, S.; PARK, Y.; CHOI, A. A survey of recommendation systems: recommendation models, techniques, and application fields. **Electronics**, MDPI, v. 11, n. 1, p. 141, 2022. Citado na página 9.
- MILETTO, E. M.; COSTALONGA, L. L.; FLORES, L. V.; FRITSCH, E. F.; PIMENTA, M. S.; VICARI, R. M. Introdução à computação musical. In: SN. **IV Congresso Brasileiro de Computação**. [S.l.], 2004. Citado 5 vezes nas páginas 6, 11, 12, 13 e 43.
- MONTEIRO, M. B.; MACHADO, N. R.; NOGUEIRA, P. A.; CAMPOS, T. A.; CRUZ, F. W.; FERNEDA, E. Um sistema de recomendação de músicas brasileiras. 2009. Citado na página 9.
- OMOHUNDRO, S. M. **Five balltree construction algorithms**. [S.l.]: International Computer Science Institute Berkeley, 1989. Citado 3 vezes nas páginas 6, 21 e 22.
- ORIO, N. et al. Music retrieval: A tutorial and review. **Foundations and Trends® in Information Retrieval**, Now Publishers, Inc., v. 1, n. 1, p. 1–90, 2006. Citado 2 vezes nas páginas 11 e 12.

PARK, D. H.; KIM, H. K.; CHOI, I. Y.; KIM, J. K. A literature review and classification of recommender systems research. **Expert systems with applications**, Elsevier, v. 39, n. 11, p. 10059–10072, 2012. Citado na página 13.

SÁ, S. P. de. Se vc gosta de madonna também vai gostar de britney! ou não? gêneros, gostos e disputa simbólica nos sistemas de recomendação musical. In: **E-Compós**. [S.l.: s.n.], 2009. v. 12, n. 2. Citado na página 9.

SINGER, P. A. M. e J. M. **Estatística e Ciência de Dados**. 2nd. ed. [S.l.]: LTC, 2013. Citado na página 25.

SONG, Y.; DIXON, S.; PEARCE, M. A survey of music recommendation systems and future perspectives. In: CITESEER. **9th international symposium on computer music modeling and retrieval**. [S.l.], 2012. v. 4, p. 395–410. Citado 4 vezes nas páginas 13, 15, 28 e 45.

SOOFI, A. A.; AWAN, A. Classification techniques in machine learning: applications and issues. **Journal of Basic & Applied Sciences**, Set Publishers, v. 13, n. 1, p. 459–465, 2017. Citado na página 16.

STELLING, R. **t-SNE Explation**. 2022. Last accessed 04 April 2024. Disponível em: <<https://observablehq.com/@robstelling/abrindo-a-caixa-preta-do-t-sne/2>>. Citado 2 vezes nas páginas 6 e 26.

THAUT, M. H. Music as therapy in early history. **Progress in brain research**, Elsevier, v. 217, p. 143–158, 2015. Citado na página 9.

WICKMEDIA. **Wikimedia KDTree**. 2024. Last accessed 06 April 2024. Disponível em: <[https://pt.wikipedia.org/wiki/%C3%81rvore\\_k-d#/media/Ficheiro:Tree\\_0001.svg](https://pt.wikipedia.org/wiki/%C3%81rvore_k-d#/media/Ficheiro:Tree_0001.svg)>. Citado 2 vezes nas páginas 6 e 23.

\_\_\_\_\_. **Wikimedia KDTree Cuts**. 2024. Last accessed 06 April 2024. Disponível em: <[https://commons.wikimedia.org/wiki/File:Kdtree\\_2d.svg](https://commons.wikimedia.org/wiki/File:Kdtree_2d.svg)>. Citado 2 vezes nas páginas 6 e 24.