PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CLASSIFICAÇÃO E RECOMENDAÇÃO DE MÚSICAS BASEADAS EM SIMILARIDADE E PROXIMIDADE COMPUTACIONAL

CARLOS HENRIQUE DE SOUZA SILVA

GOIÂNIA
2024

CARLOS HENRIQUE DE SOUZA SILVA

CLASSIFICAÇÃO E RECOMENDAÇÃO DE MÚSICAS BASEADAS EM SIMILARIDADE E PROXIMIDADE COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes da Pontifícia Universidade Católica de Goiás como parte dos requisitos para a conclusão do curso de Ciência da Computação.

Orientador(a):

Me. Gustavo Siqueira Vinhal

Banca examinadora:

Dr.ª Professor Banca1

Dr.ª Professor Banca 2

GOIÂNIA

2024

CARLOS HENRIQUE DE SOUZA SILVA

CLASSIFICAÇÃO E RECOMENDAÇÃO DE MÚSICAS BASEADAS EM SIMILARIDADE E PROXIMIDADE COMPUTACIONAL

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Ciência de Computação, em _____/_____/_____.

_____
Orientador(a): Me. Gustavo Siqueira Vinhal

_____
Prof. Me. Membro da Banca Examinadora

_____
Prof. Me. Membro da Banca Examinadora

GOIÂNIA

2024

DEDICATÓRIA

Dedico esse trabalho aos meus pais, por terem me apoiado quando escolhi este curso, aos amigos que fiz em todo o processo, pois cada um foi essencial para a conclusão dessa jornada.

AGRADECIMENTOS

Quero agradecer aos meus pais, que sempre me ensinaram o valor do estudo, e sempre me apoiaram neste e em todos os momentos da minha vida.

Ao professor Gustavo Siqueira Vinhal por ter aceitado ser meu orientador e pelo seu apoio, seus conselhos e paciência.

Aos meus amigos que fiz diretamente e indiretamente pelo processo, que foram de exímia importância para me motivar em finalizar essa jornada.

RESUMO

A música, como expressão cultural universal, desempenha um papel crucial na sociedade. Com a evolução tecnológica e o surgimento de plataformas de *streaming*, surge a necessidade de sistemas de recomendação musical mais precisos e personalizados. Este trabalho propõe a implementação de um sistema de classificação e recomendação de músicas baseado em similaridade, utilizando técnicas de proximidade computacional, algoritmos de transformação de sinais e coeficientes de sistema de reconhecimento de voz. O objetivo geral é desenvolver um sistema que analise características como tempo, BPM (Batidas por Minuto) e timbres para recomendar músicas similares às preferências do usuário. Para atingir esse objetivo, foram definidos objetivos específicos, incluindo a criação de um modelo de dados, implementação de algoritmos de pré-processamento, desenvolvimento do sistema de recomendações e validação dos resultados. Os resultados mostraram que o sistema alcançou uma taxa de acertos moderada, com variações significativas entre diferentes estilos musicais. Embora os resultados demonstrem uma taxa de acertos moderada, há espaço para melhorias, indicando a necessidade de métodos mais avançados de processamento de sinais e consideração de uma variedade maior de características musicais.

Palavras-chave: Sistemas de Recomendação, Similaridade Musical, Transformada de Fourier.

ABSTRACT

Music, as a universal cultural expression, plays a crucial role in society. With technological advancements and the emergence of streaming platforms, there is a need for more accurate and personalized music recommendation systems. This work proposes the implementation of a classification and recommendation system based on similarity, utilizing computational proximity techniques, signal transformation algorithms, and coefficients from voice recognition systems. The general objective is to develop a system that analyzes features such as tempo, BPM (Beats Per Minute), and timbre to recommend music similar to the user's preferences. To achieve this goal, specific objectives were defined, including the creation of a data model, implementation of preprocessing algorithms, development of the recommendation system, and validation of the results. The results showed that the system achieved a moderate success rate, with significant variations across different musical styles. Although the results demonstrate a moderate success rate, there is room for improvement, indicating the need for more advanced signal processing methods and consideration of a wider variety of musical features.

LISTA DE FIGURAS

Não foi encontrada nenhuma entrada de índice.

LISTA DE ABREVIAÇÕES/SIGLAS

BPM - Batidas Por minuto

CNN - Redes Neurais Convolucionais (*Convolutional Neural Network*)

SVM - Máquina de Vetor de Suporte (*Support Vector Machine*)

KNN - K Vizinhos mais próximos (K-Nearest Neighbours)

MFCC - Coeficientes Cepstrais de Função Mel (*Mel frequency Cepstral coefficients*)

Hz – Hertz (unidade de medida)

MP3 - MPEG 1 *Audio Layer* 3

MPEG - *Moving Picture Experts Group*

HDF5 - Formato de dados Hierárquicos 5 (*Hierarchical Data Format 5*)

CPU - Unidade Central de processamento (*Central Process Unit*)

MPB - Música Popular Brasileira

RAM - Memória de Acesso Randômico (*Random Access Memory*)

I/O - Entrada e Saída (I/O *Input and Output*)

SUMÁRIO

1 Introdução

A música é uma forma de arte que transcende fronteiras culturais, linguísticas e temporais. Desde sempre, ela tem sido uma expressão vital da identidade, emoção e criatividade dos povos. Com sua capacidade única de evocar sentimentos, transmitir narrativas e conectar pessoas, a música desempenha um papel fundamental em todas as sociedades. Despertando sensações muitas vezes individuais, a música ao longo dos séculos tem sido um poderoso veículo de expressão humana. De fato, o gosto musical é tão diverso e pessoal quanto às experiências de vida de cada pessoa.

No entanto, com o advento da tecnologia e o surgimento de novas maneiras de consumi-la, como as plataformas de streaming, e com tantas opções agora disponíveis, surge o questionamento sobre o que ouvir a seguir. Neste contexto, surge a necessidade de sistemas de recomendação musical mais eficazes e personalizados, capazes de levar em consideração a individualidade do gosto musical de cada usuário.

É comum que ouvintes de música busquem canções similares às que já apreciam, entretanto, frequentemente encontram dificuldades em localizar exatamente o que desejam. Este desafio é particularmente evidente ao utilizar modos de reprodução aleatória, onde a descoberta de novas músicas que correspondam ao gosto pessoal pode ser um processo árduo e desorientador. Muitas vezes, os usuários não sabem por onde começar a procurar, o que evidencia a necessidade de sistemas de recomendação mais eficazes e intuitivos.

A importância de sistemas de recomendação musical eficazes vai além da experiência do usuário. Eles têm potencial para:

1. **Apoiar Artistas Emergentes**: Sistemas bem desenhados podem ajudar novos artistas a encontrar seu público-alvo, permitindo que talentos emergentes sejam descobertos e valorizados.
2. **Melhorar a Experiência do Usuário**: Usuários se beneficiam de recomendações precisas e personalizadas que ampliam seu repertório musical e enriquecem suas experiências auditivas.

3. **Impulsionar a Indústria Musical**: Recomendadores eficazes podem aumentar o engajamento dos usuários e, consequentemente, o consumo de música, beneficiando financeiramente tanto as plataformas quanto os artistas.

Este trabalho propõe-se a estudar e explorar formas de classificar e recomendar músicas utilizando técnicas avançadas de processamento de áudio e algoritmos de análise de proximidade e similaridade. Por meio dessas abordagens, busca-se desenvolver um sistema capaz de identificar características musicais distintas e agrupá-las de maneira a proporcionar recomendações mais precisas e personalizadas para os usuários. Portanto a questão chave a ser respondida é:

É possível, por meio da utilização de processamento computacional, descobrir e agrupar músicas com características similares?

1.1 Objetivo Geral

O objetivo geral deste trabalho é implementar um sistema de classificação e recomendação de músicas baseado em similaridade. Será analisado será desde tempo, BPM (batidas por minuto), timbres e outros coeficientes previamente calculados.

1.2 Objetivo Específico

- Criar um modelo de dados próprio
- Implementar um algoritmo de pré-processamento e otimização de dados
- Desenvolver um sistema de recomendações baseados nos dados obtidos
- Realizar testes e validação com opinião popular
- Documentar e apresentar dos resultados

2 Referencial Teórico

Neste capítulo serão expostos os conceitos principais dos quais permeiam a execução deste trabalho a estrutura do referencial teórico nesta pesquisa foi dividida em: Trabalhos relacionados, Coeficientes e Similaridade de Cosseno.

2.1 Trabalhos Relacionados

Para Silla et al (2008), a música é hoje uma parte significativa do conteúdo da Internet. A internet é, provavelmente, a fonte mais importante de músicas, com vários sites dedicados à disseminação, distribuição e comercialização da música. Dentre esses sites existem plataformas que possuem milhões de acessos e equivalentemente milhões de músicas em seus acervos. Para essas plataformas, de acordo com Elbir (2020), classificar, agrupar, nomear e recomendar essas músicas de forma correta é um dos pilares que melhoram suas receitas e sua aceitação pelo público. A maioria dessas plataformas oferecem o serviço de recomendação baseado em filtragem colaborativa e análise de metadados.

Na área de classificação de gêneros, diversos métodos já foram sugeridos, e com eles uma infinidade de técnicas. Alguns pesquisadores como Bahuleyan (2018) e Xu et al (2005) compararam a eficiência do treinamento de modelos computacionais de CNN (*Convolutional Neural Network* - Redes Neurais Convolucionais). É um sistema feito de coeficientes calculados sobre o domínio do tempo e o domínio da frequência para identificar qual seria mais eficiente utilizando aprendizagem de máquina convencional como regressão logística e florestas randômicas.

Ndou et al (2021), utilizando um modelo padrão denominado GTZAN apresentam similarmente diversos dados que podem ser extraídos de uma amostra de áudio de tamanho fixo e os emprega no treinamento de diversos algoritmos de aprendizagem de máquina como por exemplo: SVM (*Support Vector Machine* – Máquina de Vetor de Suporte), KNN (*K-Nearest Neighbours* – K Vizinhos mais próximos), CNN. Nesta pesquisa os autores apresentam uma visão ampla dos resultados, classificando-os dentre os que se saem melhor, trazendo um resultado

mais preciso. Os testes dos modelos são feitos com diversos conjuntos desses dados, para também identificar quais dos coeficientes mais influenciam no resultado

Além dessas técnicas para classificação, Silla et al (2008) apresenta uma forma de otimização de processamento, gerando uma média ponderada dos sinais, descartando as partes da música onde se identifica o silêncio para aumentar a eficácia e diminuir o tempo de processamento e o uso de memória alocada.

Deste ponto de vista, é possível vetorizar os dados extraídos de um sinal de áudio em coeficientes que são utilizados para o processamento de redes neurais. Ndou et al (2021) cita os coeficientes baseados em magnitude, coeficientes baseados em tempo, coeficientes baseados no tom e progressão de acordes.

Além disso Xu et al (2005) apresenta uma forma de melhorar a classificação dos componentes do vetor particionando a banda de frequência do áudio de 0 hz – 22025 hz em 10 sub bandas de frequência e, em seguida, agrupando essas bandas para geração do determinado coeficiente.

2.2 Coeficientes

A extração dos dados importantes para que o sistema possa operar também é um ponto importante da pesquisa. Uma das principais métricas utilizadas para se categorizar e realizar reconhecimento de áudio são os Coeficientes Cepstrais de Função Mel (MFCC - *Mel frequency Cepstral coefficients*).

2.2.1 MFCC

A MFCC é uma técnica usada para extrair características importantes do áudio, principalmente fala, mas também pode ser usada para sons musicais. O processo de geração dos coeficientes pode ser descrito, de acordo com Gupta et al (2013), com o fluxograma descrito na Figura 01.

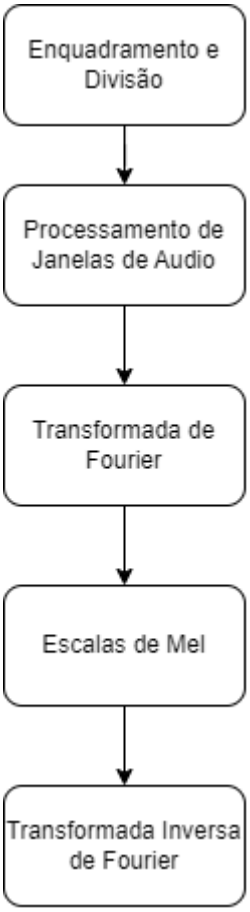

Figura 1 - Fluxograma passo-a-passo do processo de processamento dos coeficientes MFCC. Fonte: Autoria Própria.

De acordo com o fluxograma, as etapas da extração são: Enquadramento e Divisão, Processamento de janelas de áudio, Transformada de Fourier, Escalas de Mel e Transformada Inversa de Fourier.

2.2.1.1 Enquadramento e Divisão

Segundo Gupta et al (2013) na etapa de Enquadramento e Divisão, o sinal de áudio é dividido em blocos e em seguida separado em quadros para seu processamento ser realizado. O tamanho desses blocos e quadros deve ser ajustado de forma que caso os quadros sejam muito pequenos, é possível não conseguir extrair nenhuma informação útil, e caso o quadro seja muito grande, as informações podem sofrem muita variação tornando assim precária sua utilização. Os blocos devem ser

divididos de forma que permitam o processamento mais eficiente do sinal. Esse processo se repete durante todo o sinal.

2.2.1.2 Processamento das janelas do áudio

Nesta etapa Gupta et al (2013) descreve a minimização das perturbações no início e no fim de cada quadro. Além disso, essa etapa também é responsável para que cada amostra do sinal tenha a mesma quantidade de quadros.

2.2.1.3 Transformada de Fourier

A transformada de Fourier é uma ferramenta essencial do processamento de sinais e na matemática que permite dividir um sinal ou uma função em suas frequências constituintes. É uma técnica eficaz para a análise e representação de sinais no domínio da frequência, o que é essencial em várias áreas, como engenharia elétrica, física e análise de dados. Esta etapa é responsável por transformar o sinal recebido para o domínio da frequência e ser utilizado na etapa posterior. Para melhor otimização computacional é utilizado a Transformada Rápida de Fourier, devido seu menor custo e seu resultado ser o mesmo pela Transformada Discreta de Fourier (GUPTA, 2013).

2.2.1.4 Escalas de Mel

Com a transformada recebida anteriormente, é realizado o mapeamento dos espectros de áudio de acordo com a escala de Mel para se identificar a energia existente em cada ponto (GUPTA, 2013). Essa escala é utilizada para se decompor as frequências em amostras que são mais relevantes para o ouvido humano. De início, as frequências mais baixas são utilizadas por um filtro linear. A partir de 1000 Hz o filtro se torna logaritmo de forma que sua curva filtra mais os espectros cuja banda é mais aguda. A fórmula para o cálculo pode ser descrita de acordo com a Equação 01.

$$M_f = 2595(\frac{f}{700} + 1)$$

Equação 1 – Cálculo de Mel

Aplicando a Equação 1 para os diferentes valores de frequência, é possível obter o gráfico representado pela Figura 2.

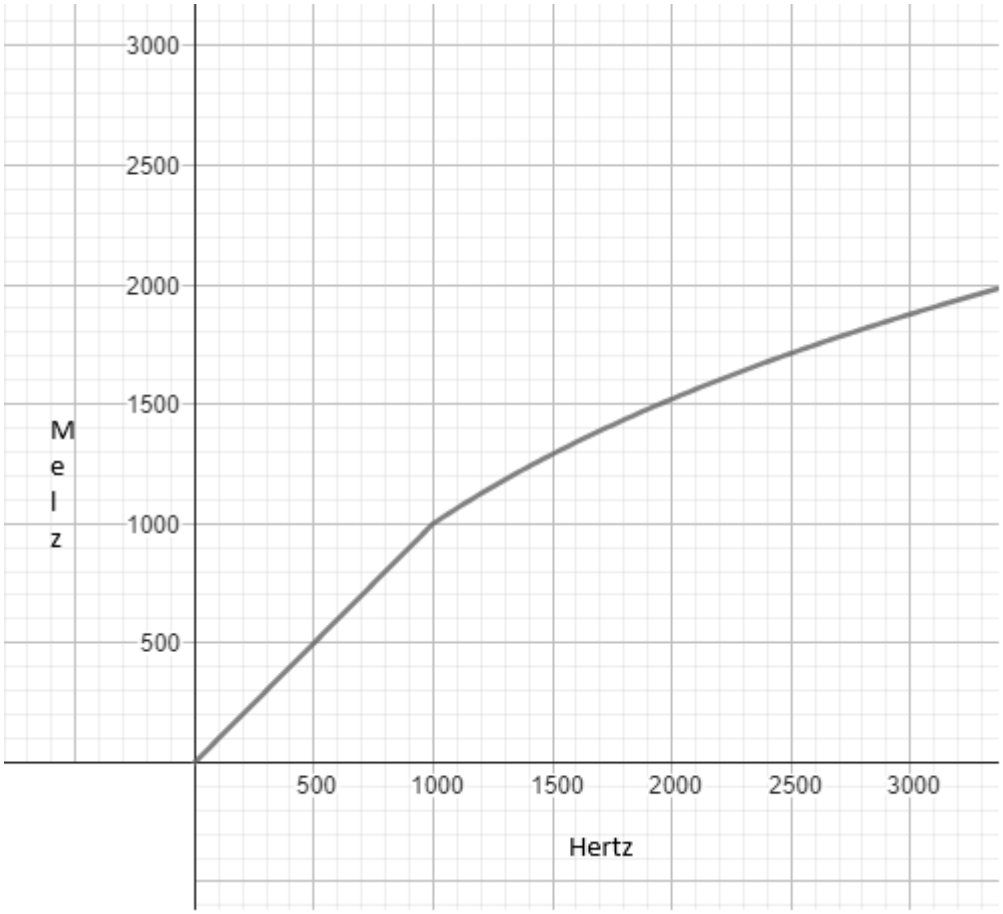

Figura 2 - Gráfico da mudança dos pesos considerados pelo cálculo da MFCC
Fonte: Autoria Própria.

De acordo com a Figura 2, é possível observar que até 1000 Hz os valores de Mel seguem lineares. Após isso, o crescimento se torna mais devagar.

2.2.1.5 Transformada Inversa de Fourier

Esta etapa apenas transforma novamente a matriz de magnitude gerada pela escala de Mel para que seja possível a obtenção dos coeficientes de Mel. Para Gupta

et al (2013), os primeiros 13 foram utilizados em seu algoritmo e trouxeram resultados sem muitos ruídos.

2.3 Similaridade de Cosseno

Similaridade é um conceito essencial para classificação de dados. É por meio dela que se pode verificar quão perto dois conjuntos estão entre si. A similaridade de cosseno corresponde a esse conceito. É possível calcular quão próximos dois vetores em um espaço vetorial estão baseados em seus ângulos de índice.

Além disso, a similaridade tende a ser inversamente proporcional à distância euclidiana. Porém esse cálculo é muito sensível para grandes magnitudes tornando os resultados muito imprecisos para qualquer variação mínima da entrada.

Por outro lado, a similaridade de cosseno segue sua variação pelo ângulo dos vetores no espaço, sendo muito mais tolerante com grande magnitude de dados (XIA, 2015). Porém esse cálculo é sensível à orientação do vetor de dados, sendo necessário realizar alguns ajustes quanto ao módulo das entradas, de forma a manter os vetores na mesma orientação.

2.4 Teoria Musical

A teoria musical é o estudo dos elementos de música, incluindo melodia, harmonia, ritmo, e estrutura. Compreender esses elementos é fundamental para o desenvolvimento de sistemas de recomendação musical, pois eles influenciam diretamente a maneira como a música é percebida e apreciada pelos ouvintes. Para Dantas et. al. (2019) Ela aborda como os sons são produzidos, combinados e representados, tanto do ponto de vista musical quanto físico, evidenciando a interseção entre a representação matemática dos fenômenos físicos e a escrita musical .

2.4.1 Melodia

A melodia é uma sequência de notas que são percebidas como uma única entidade. Ela é a parte da música que é mais frequentemente lembrada e cantada. A

análise de melodias pode incluir a identificação de intervalos, escalas, e frases musicais. (Dantas et. al. 2019)

2.4.2 Harmonia

A harmonia, segundo Dantas et. al. (2019), refere-se à combinação de diferentes notas tocadas ou cantadas simultaneamente. Envolve a formação de acordes, consonâncias e dissonâncias, e como esses acordes se organizam para acompanhar e harmonizar melodias. A harmonia explora a superposição de ondas sonoras que ocorrem quando várias notas são tocadas ao mesmo tempo A harmonia dá profundidade e contexto à melodia e pode influenciar a emoção e o caráter da música.

2.4.3 Ritmo

O ritmo é organização temporal dos sons e silêncios na música, criando a sensação de movimento no tempo. (Dantas et. al. 2019) Ele inclui elementos como tempo (BPM - batidas por minuto), assinatura de tempo, e subdivisões rítmicas. O ritmo é crucial para a identificação de gêneros musicais e para a sincronização de elementos musicais em sistemas de recomendação.

2.4.4 Timbre

O timbre é a qualidade do som, é o que diferencia dois sons que têm a mesma altura e intensidade. Ele é influenciado pela forma de onda do som e pelos harmônicos presentes. A análise do timbre é importante para a classificação de músicas baseadas em similaridade. Ele é determinado pela composição espectral da onda sonora e é influenciado pelos harmônicos gerados pela fonte sonora, permitindo distinguir entre diferentes instrumentos musicais ou vozes. (Dantas et. al. 2019)

A teoria musical fornece uma base sólida para a construção de sistemas de recomendação musical. Ao analisar e comparar elementos como melodia, harmonia, ritmo e timbre, os sistemas podem oferecer recomendações personalizadas que correspondem às preferências musicais dos usuários. Essas definições evidenciam a

relação entre a linguagem musical e a física, mostrando como ambos os campos se complementam na representação e entendimento dos fenômenos sonoros

3 Metodologia de Pesquisa

Antes da especificação dos métodos e atividades realizadas é importante destacar os limites dessa pesquisa. Conforme as conceituações oferecidas por Gil (2002) e Wazlawick(2014) esse trabalho tem característica de resumo de assunto, uma vez que serão utilizados conceitos já presentes na comunidade científica, onde o principal ponto é a aplicação de diversos conceitos com um objetivo em comum. Quanto aos procedimentos técnicos, a classificação compõe uma pesquisa experimental e exploratória.

Na pesquisa experimental, de acordo com Wazlawick(2014), os pesquisadores têm atuação direta sobre os objetos de estudo, alterando parâmetros, instâncias e técnicas, bem como avaliando os resultados alcançados para verificar a validade das hipóteses estabelecidas.

Por outro lado, Gil (2002) descreve a pesquisa exploratória de forma a buscar compreender fenômenos pouco conhecidos ou explorados, proporcionando uma visão inicial sobre o tema em questão, sem a pretensão de esgotar todas as possibilidades de análise. Dessa forma, a presente pesquisa adota uma abordagem mista, combinando elementos da pesquisa experimental para avaliação de resultados e da pesquisa exploratória para investigação dos fenômenos explorados.

4 Desenvolvimento

Neste capítulo serão detalhadas as ferramentas necessárias para o desenvolvimento do trabalho.

4.1 Python

Para o desenvolvimento deste trabalho, a escolha da a linguagem Python se justifica devido sua versatilidade e capacidade de lidar com grande volume de dados, além de sua curva de aprendizado ser baixa e coesa. De acordo com sua documentação (Python.org, 2024), a linguagem Python é conhecida por sua sintaxe limpa e fácil de ler, que se assemelha à linguagem humana. Isso o torna de amplo acesso e fácil leitura e manutenção. Além disso, pode ser usada para uma ampla variedade de tarefas, incluindo desenvolvimento web, ciência de dados, *machine learning*, automação de tarefas e entre outros. Outro ponto decisivo é que sua licença é código aberto, o que significa que é gratuita para usar e modificar. Isso a torna uma opção acessível.

A coleção de bibliotecas públicas disponível em Python é bastante vasta, o que permite utilizar algoritmos que são referência na comunidade, extensamente testados e validados. A escolha da linguagem propriamente dita no tópico anterior tem seu pilar baseado nessa capacidade. Desta forma várias bibliotecas foram escolhidas para o tratamento e execução dos algoritmos, sendo elas:

- **os:** o módulo “os” fornece funcionalidades para interagir com o sistema operacional subjacente. Ele é amplamente utilizado para manipulação de arquivos e diretórios, gerenciamento de processos, manipulação de caminhos de arquivo e obtenção de informações do ambiente de execução. Funções como os.listdir() e os.path.join() são comumente usadas para listar arquivos em diretórios e criar caminhos de arquivo de forma independente da plataforma, respectivamente.
- **eyed3:** a biblioteca eyed3 é uma ferramenta para trabalhar com arquivos de áudio em formato MP3. Os desenvolvedores podem usar o eyed3 para extrair e alterar metadados de arquivos MP3, que incluem informações como título,

artista, álbum e ano de lançamento da música. Essa funcionalidade é essencial para aplicações que visam organizar e analisar bibliotecas de música digital, pois fornece um conjunto robusto de recursos para tornar esses processos mais rápidos e precisos.

- **re:** o módulo re do Python suporta expressões regulares, o que é uma ferramenta poderosa para tratamento de cadeias de caracteres (*strings*). As expressões regulares são padrões criados para encontrar correspondências particulares em frases. O módulo re permite que os desenvolvedores realizem uma variedade de tarefas complexas, como busca, correspondência, substituição e divisão de strings, tudo com base em padrões específicos. Essa funcionalidade é essencial para a manipulação de texto complexa em Python, pois permite a criação de aplicações mais adaptáveis e eficientes.

- **time:** o módulo time é usado para trabalhar com valores de tempo e realizar operações relacionadas ao tempo. Ele oferece funções para medir o tempo de execução de um trecho de código, pausar a execução por um determinado período e obter a hora atual. O time é fundamental para aplicações que envolvem agendamento, temporização e medição de desempenho.

- **concurrent.futures:** o módulo concurrent.futures fornece uma interface de alto nível para a execução de código concorrente em Python. Para maximizar os recursos do sistema operacional, essa ferramenta permite a paralelização de tarefas por meio de threads ou processos. Esse método é especialmente útil para tarefas que podem ser executadas de forma independente e se beneficiam da execução simultânea. Os desenvolvedores podem usar o concurrent.futures para melhorar o desempenho de seus programas em ambientes com vantagens de concorrência.

- **numpy:** NumPy é uma biblioteca fundamental para computação científica em Python. Ela fornece suporte para arrays multidimensionais, alongamento de funções matemáticas, álgebra linear, transformada de Fourier e outras

operações matemáticas eficientes. NumPy é amplamente utilizado em áreas como ciência de dados, processamento de sinais e computação numérica.

- **librosa:** a biblioteca librosa fornece uma variedade de recursos para processamento de sinais de áudio, tornando-se uma ferramenta indispensável para análise de áudio em Python. Além de extrair características e manipular sinais, a librosa é altamente valorizada por sua capacidade de calcular representações espectrais e espectrogramas complexos. O reconhecimento de fala, a análise musical e até mesmo a pesquisa científica com dados de áudio são algumas das muitas aplicações que o tornam uma escolha popular. A biblioteca ainda é uma parte importante do ecossistema de processamento de áudio em Python devido à sua extensa documentação e a uma comunidade de usuários ativa. Isso permite que os desenvolvedores explorem e extraiam facilmente dados valiosos de arquivos de áudio.

- **h5py:** a biblioteca h5py fornece uma interface Python para o formato de arquivo de dados HDF5. Ela permite armazenar e manipular grandes conjuntos de dados de forma eficiente. O h5py é comumente usado em aplicações que lidam com conjuntos de dados volumosos, como aprendizado de máquina e processamento de sinais.

- **pydub:** a biblioteca pydub é uma ferramenta poderosa para manipulação de áudio em Python. Ela oferece suporte para operações como corte, mistura, conversão de formato e entre outros. O pydub é amplamente utilizado em aplicações de edição de áudio, processamento de sinais e análise de áudio.

- **multiprocessing:** o módulo multiprocessing permite a execução de tarefas em paralelo usando processos em vez de threads. Ele é especialmente útil para tarefas intensivas em CPU que podem se beneficiar da execução paralela. O multiprocessing é amplamente utilizado em aplicações que precisam aproveitar ao máximo os recursos de hardware disponíveis.

- **sklearn:** Scikit-learn, também conhecida como sklearn, é uma biblioteca de aprendizado de máquina em Python que oferece uma ampla gama de ferramentas eficientes e fáceis de usar para análise de dados e modelagem preditiva. Desenvolvida como uma extensão do SciPy, a biblioteca foi lançada inicialmente em 2007 por David Cournapeau como parte de um projeto de verão do Google. Desde então, ela cresceu significativamente e agora é mantida por uma comunidade ativa de desenvolvedores e cientistas de dados, incluindo grandes contribuições de Olivier Grisel, Gael Varoquaux e muitos outros.

Dentre as bibliotecas citadas, a Scikit Learn desempenha um papel crucial no desenvolvimento do algoritmo de recomendação. A biblioteca oferece uma ampla gama de funcionalidades que cobrem praticamente todos os aspectos do aprendizado de máquina supervisionado e não supervisionado. Suas principais vantagens incluem:

- **Facilidade de Uso**: Scikit-learn possui uma API consistente e documentada, que facilita a aprendizagem e o uso.
- **Desempenho**: A biblioteca é altamente otimizada, permitindo manipulação eficiente de grandes volumes de dados.
- **Integração**: Seu funcionamento com outras bibliotecas do ecossistema Python, como NumPy, Pandas e Matplotlib, facilita a integração em fluxos de trabalho de processamento de dados.

Uma das muitas ferramentas oferecidas pelo Scikit-learn é a função cosine_similarity. Esta função é usada para medir a similaridade entre dois vetores em um espaço multidimensional. A similaridade cosseno é uma medida que calcula o cosseno do ângulo entre dois vetores, fornecendo um valor entre -1 e 1, onde 1 indica vetores idênticos e -1 indica vetores completamente opostos.

A similaridade cosseno é particularmente útil em tarefas como:

- **Análise de Textos**: Para medir a similaridade entre documentos baseados na frequência de palavras.

- **Sistemas de Recomendação**: Para encontrar itens semelhantes baseados nas preferências dos usuários.
- **Agrupamento de Dados**: Para identificar padrões e agrupar dados similares em clusters.

4.2 Modelo de Dados

O modelo de dados para esta pesquisa foi criado após uma análise de modelos semelhantes em trabalhos anteriores. Muitos desses modelos usavam uma estrutura de cerca de cem músicas organizadas em dez gêneros diferentes. Mas havia preocupações com essa abordagem porque poderia prejudicar a eficácia das recomendações, especialmente pelo objetivo de gerar uma lista contendo ao menos dez itens relevantes. Como resultado, o desenvolvimento de um modelo mais abrangente e representativo tornou-se obrigatório. Esse modelo deveria ser capaz de lidar com um número maior de dados e fornecer recomendações mais diversificadas e precisas aos usuários.

Assim, para melhorar a qualidade e a relevância das recomendações apresentadas, um novo modelo foi desenvolvido levando em consideração variáveis, como características musicais, preferências do usuário e tendências de mercado. Para melhorar a experiência de recomendação dos usuários e aumentar sua satisfação, essa abordagem mais abrangente e personalizada foi empregada.

Com o apoio do projeto everynoise.com, que compila uma extensa lista de dados rastreados e analisados, contendo 6.291 distintas classificações de gêneros do Spotify até 19 de novembro de 2023 (McDonald, Gleen 2013-2023), foi possível selecionar 34 gêneros principais. Esses gêneros foram cuidadosamente escolhidos para garantir uma representação abrangente da diversidade musical, abrangendo desde estilos clássicos até tendências contemporâneas. Os gêneros selecionados incluem uma variedade de estilos como ambient, blues, chillwave, country, cyberpunk, disco, djent, downtempo, drill, dub, emo, experimental, folk, forró, funk, future bass, glitch, hip hop, house, jazz, kpop, lo fi, metal, MPB, orchestra, phonk, pixel, pop, punk, rap, reggae, rock, soul e trap, cada um contendo em média 100 músicas. Essa ampla

diversidade de gêneros e o tamanho substancial do conjunto de dados garantem que as recomendações geradas possam abranger uma ampla gama de preferências musicais dos usuários, indo além das limitações de base de similaridade e proporcionando uma experiência de recomendação mais rica e personalizada. A Figura 3 apresenta os estilos musicais utilizados.

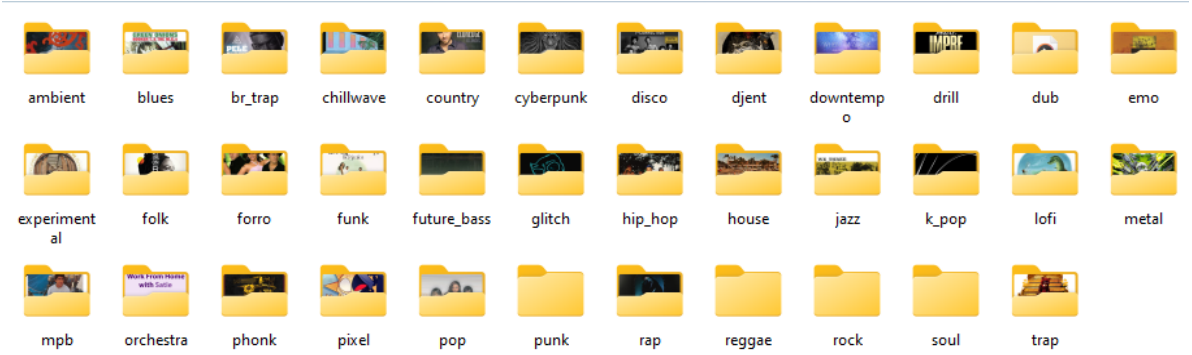

Figura 3 – lista de pastas de arquivos contendo o modelo de dados, com 35 pastas e 100 arquivos por pasta. Fonte: Autoria Própria.

4.3 Método de extração de coeficientes

O cálculo e extração dos coeficientes é a etapa principal do processamento das músicas. Nesta era lido os metadados dos arquivos de áudio para que fosse identificado seu nome, o artista e seu gênero, que previamente foi categorizado como apresenta o fluxograma da Figura 4. Essas informações são essenciais para que seja possível apresentar a lista de recomendações de similaridade. Neste ponto, de forma a otimizar o processo e evitar retrabalho, todos os arquivos foram reescritos para conter essas informações.

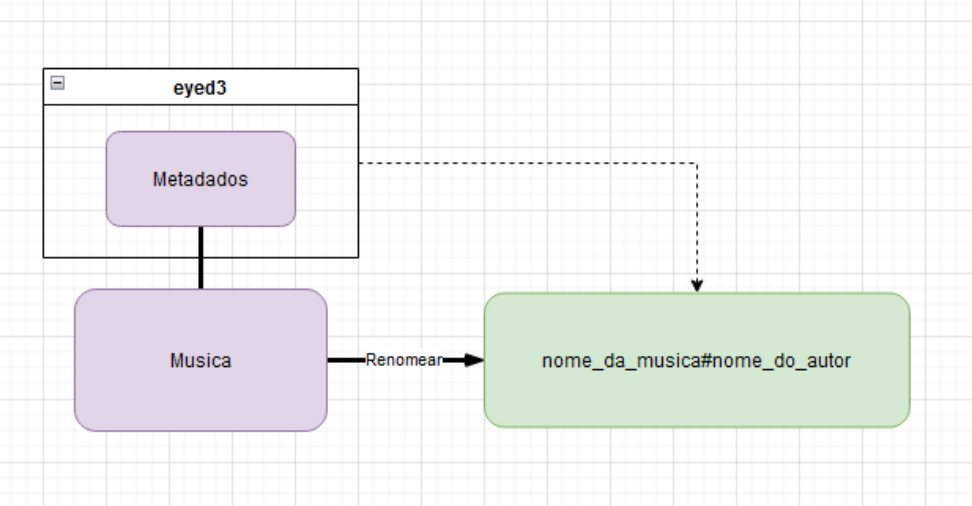

Figura 4 – Fluxograma do processo de renomeação dos arquivos com o eyed3.
Fonte: Autoria Própria.

Para isso se justifica a utilização da biblioteca eyed3, devido a sua utilidade de leitura de metadados conforme a função “renomear_arquivos_mp3” apresentada no apêndice B.

Os arquivos processados receberam a seguinte estrutura: *nome_da_musica#nome_do_artista.mp3* , desta forma se torna possível que o processamento seja realizado apenas uma vez, e facilita buscar esse dado novamente caso necessário posteriormente.

Outro ponto utilizado para melhoria da performance do algoritmo, foi adaptar uma estratégia de processamento paralelo conforme a Figura 5. Nesta etapa, foi utilizado a estratégia de multi-threads, devido ao alto índice de ações de entrada e saída do sistema operacional, pois as etapas de leitura e escrita dos arquivos no disco tomariam todo o tempo disponível. Pode ser observado pela função “pr_nomes” presente no apêndice B.

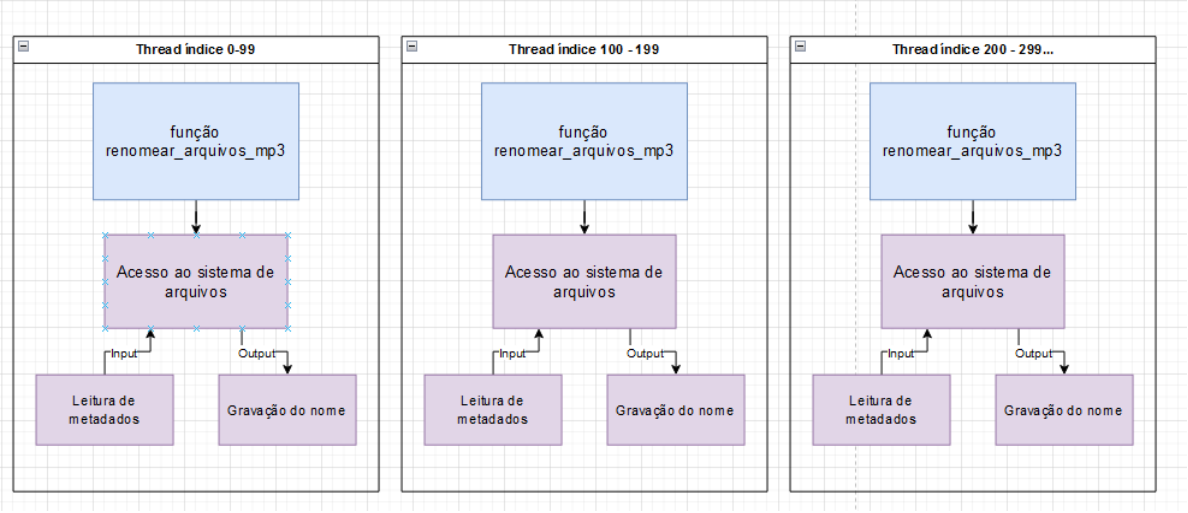

Figura 5 – Fluxograma de paralelização da execução de Threads para otimizar a entrada e saída de dados. Fonte: Autoria Própria.

Utilizando-se a biblioteca de concurrent.futures, com o máximo de threads que o sistema pudesse lidar, o tempo de execução foi de aproximadamente 50 minutos.

O processamento dos coeficientes foi feito de acordo com o processo já conhecido para obtenção listado nos trabalhos de Ndou et al (2021), Bahuleyan (2018), Silla et al (2008) e Xu et al (2005), e pode ser observado pela classe “AudioDataProcess” presente no apêndice, transformando o áudio em domínio da frequência com uso da Transformada de fourier e calculando além da MFCC os demais coeficientes que a biblioteca librosa permitiu-se extrair

Nesta etapa do processamento a matriz gerada estava concatenada de forma horizontal, o que dificultava uma precisão no algoritmo, portanto foi necessário calcular a sua transposição, para melhor organização dos dados. Além disso foi preciso também a remoção dos intervalos de silêncio no início e fim dos arquivos, de forma também a melhorar a convergência do algoritmo de recomendação.

Por fim, a função “pr_musica” do apêndice B adiciona o nome, autor e gênero ao vetor de dados para que possa ser utilizado no fim do processo de recomendação

A matriz de coeficientes ficou dividida em janelas de 256 partes com 2048 quadros cada, o que corresponde a cerca de 10 milissegundos de áudio, portanto

cada matriz possui em média entre 40 mil e 60 mil linhas de 56 colunas cada. o resultado tinha um consumo de memória de aproximadamente 600 megabytes, e com cerca de 20 músicas processadas haveria um estouro de memória RAM na maioria dos computadores populares (cerca de 16 gigabytes de memória RAM). Diante disso foi realizado uma média dos valores como Silla et al (2008) mostra em seu trabalho, para que viesse a ser possível processar todos os arquivos.

O processamento dos coeficientes utiliza também uma estratégia de processamento paralelo conforme a função “pre_prc” presente no apêndice B mostra, porém como o uso de CPU agora é maior que o uso de I/O, foram utilizados processos para realizar esse paralelismo, como apresentado a seguir pela Figura 6.

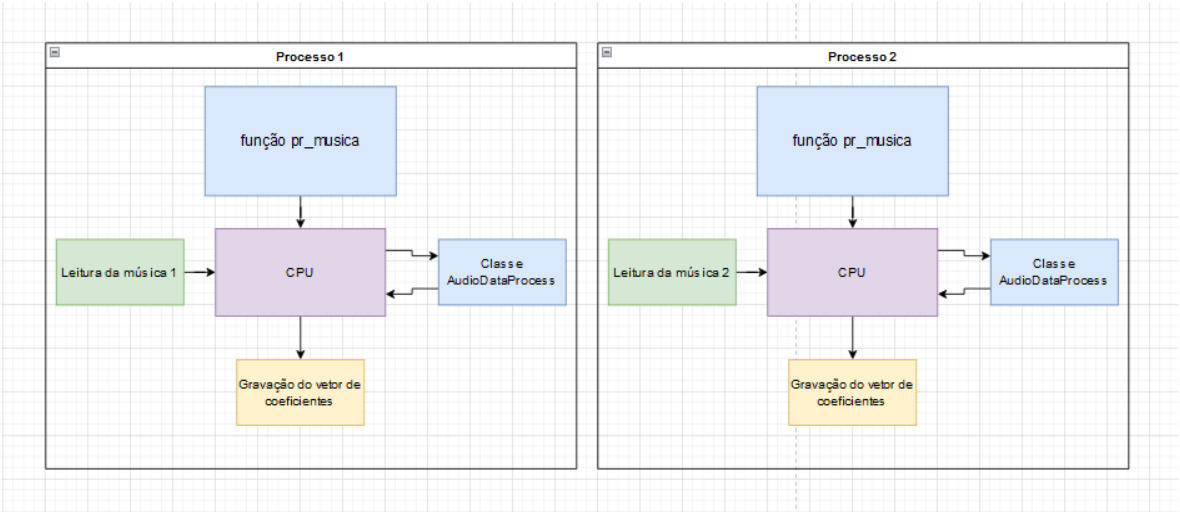

Figura 6 – Fluxograma da paralelização do processamento dos vetores de coeficiente para maximizar o uso de núcleos. Fonte: Autoria Própria.

O tempo de processamento por arquivo antes do processamento paralelo era em média de 25 segundos, totalizando mais de 20 horas, porém agora com mais de um processo em execução pelo sistema, é possível executar N músicas simultaneamente. Para o equipamento utilizado, foi disponibilizado um processador com 4 núcleos, portanto a redução foi aproximadamente 4 vezes o tempo original, chegando em torno de 6 horas totais de processamento.

Os coeficientes gerados por meio deste processo estão diretamente ligados às características estruturais e rítmicas da música, permitindo uma análise detalhada dos

elementos que compõem a base rítmica de uma faixa musical. Esses coeficientes incluem cálculos de centroides, que ajudam a identificar o centro de massa das frequências presentes em um determinado segmento da música, e a distribuição de valores ao longo de um espectrograma, que proporciona uma visualização clara de como as frequências variam ao longo do tempo. Além disso, a modulação, que mede as variações na amplitude e frequência, é analisada para entender melhor a dinâmica rítmica da música.

É importante salientar que os aspectos melódicos e vocais apresentam menor compatibilidade com este método devido à sua natureza complexa e variada. A melodia, sendo a sequência de notas que é percebida como uma única entidade, possui nuances e variações que não são capturadas completamente pelos coeficientes baseados em análise rítmica e estrutural. Da mesma forma, os elementos vocais, que incluem a prosódia, timbre e articulação, possuem uma riqueza e diversidade que desafiam a simplificação necessária para este tipo de análise.

4.4 Armazenamento

O processamento de coeficientes apresentado anteriormente, mesmo utilizando de estratégias de melhoria de performance, apresenta ainda um tempo de execução muito alto. Cada iteração em todos os arquivos do modelo custa em torno de 6 horas da execução ininterrupta do algoritmo, o que dificultaria a realização de testes de pequenas alterações de fluxo. Uma forma para solucionar este problema foi através do armazenamento dos coeficientes calculados em disco, ou seja, ao realizar o processamento de um valor, ele é salvo na forma em que seu vetor de coeficiente é gerado pelo algoritmo por meio da função “save_ftv” do apêndice B.

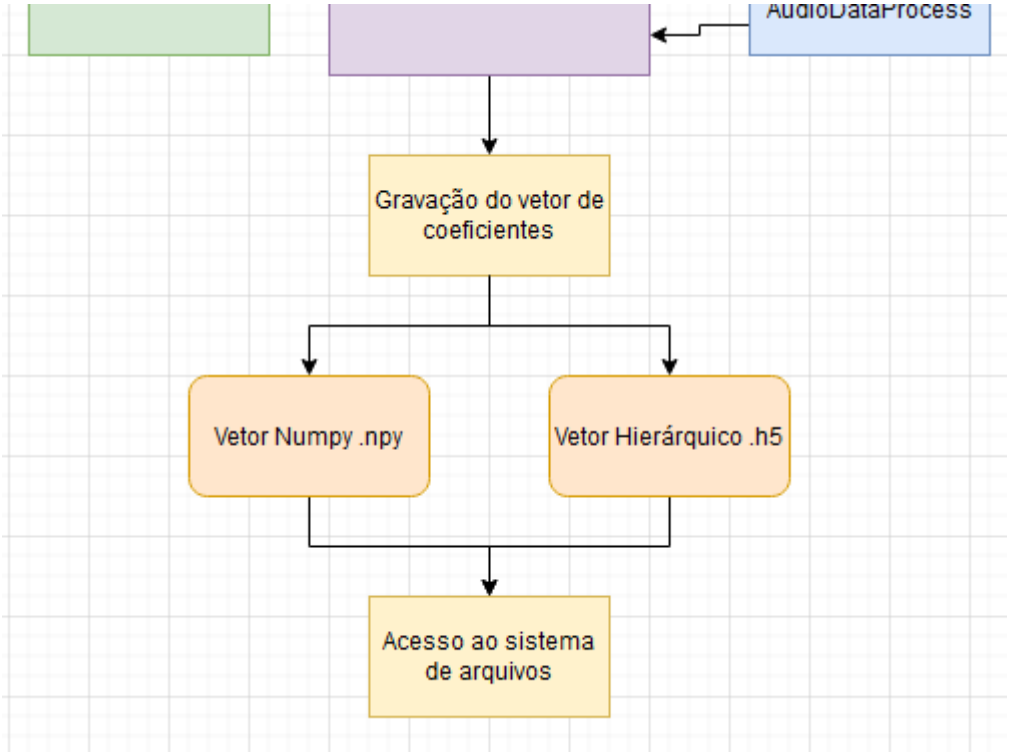

Figura 7 – Fluxograma do processo de gravação dos vetores de coeficientes e duas formas possíveis (.npy e .h5) Fonte: Autoria Própria.

Para salvar esses coeficientes, dois métodos de gravação foram testados, como exemplifica o fluxograma da Figura 7, sendo o método *npy* do qual é gerenciado pela biblioteca numpy, e o método *h5py* do qual é gerido pela biblioteca que leva seu nome: h5py. Ambos os métodos são semelhantes em seus testes, onde o arquivo salvo possuía tamanho semelhante e o tempo de gravação e leitura eram equivalentes.

A biblioteca h5py oferece maior suporte à volumetria de dados maiores, além de permitir a leitura de partes específicas do dado salvo, não precisando carregar inteiramente em memória, por isso foi escolhido em primeiro momento para o salvamento dos vetores completos. Cada vetor tinha em média 600 Megabytes, fazendo o modelo completo ter aproximadamente 140 Gigabytes de dados. A leitura de todos os vetores ainda tomava 50 minutos, porém uma redução de mais de 6 vezes foi observada se tratando do processamento sem o método de salvamento.

Porém a leitura de 140 Gigabytes de dados apresentaria erro quando alocado em memória, pois o equipamento utilizado para o processamento continha um total

de 40 Gigabytes de memória RAM, por esse motivo foi utilizado o método de média dos vetores, que reduziu os vetores salvos para arquivos de cerca de 10 kb, totalizando menos de 40 Megabytes de utilização de RAM. Para esses arquivos o npy foi utilizado devido sua facilidade de implementação.

4.5 Processamento

O processamento dos dados foi realizado por meio da similaridade de cosseno conforme a função “calculate_similarity” presente no apêndice B. Realizando a leitura dos dados salvos na etapa anterior, é carregado todos os vetores na memória e em seguida aplicado o processo de similaridade por meio da biblioteca Scikit Learn.

O cálculo da similaridade de cosseno era realizado retornando a matriz de similaridade contendo cada elemento e a proximidade do mesmo em relação ao vetor total, desta forma, vetores de coeficientes parecidos estariam próximos em relação ao espaço vetorial.

Em seguida foi aplicado o algoritmo de recomendação como a função “generate_recommendations” presente no apêndice B. Com a matriz de similaridade gerada, o proximo passo é identificar o vetor desejado e em seguida executar uma ordenação baseada neste vetor, para buscar todos os vizinhos próximos no espaço vetorial.

O algoritmo retorna as 10 primeiras posições mais próximas de acordo com o índice do vetor escolhido conforme a Figura 8.

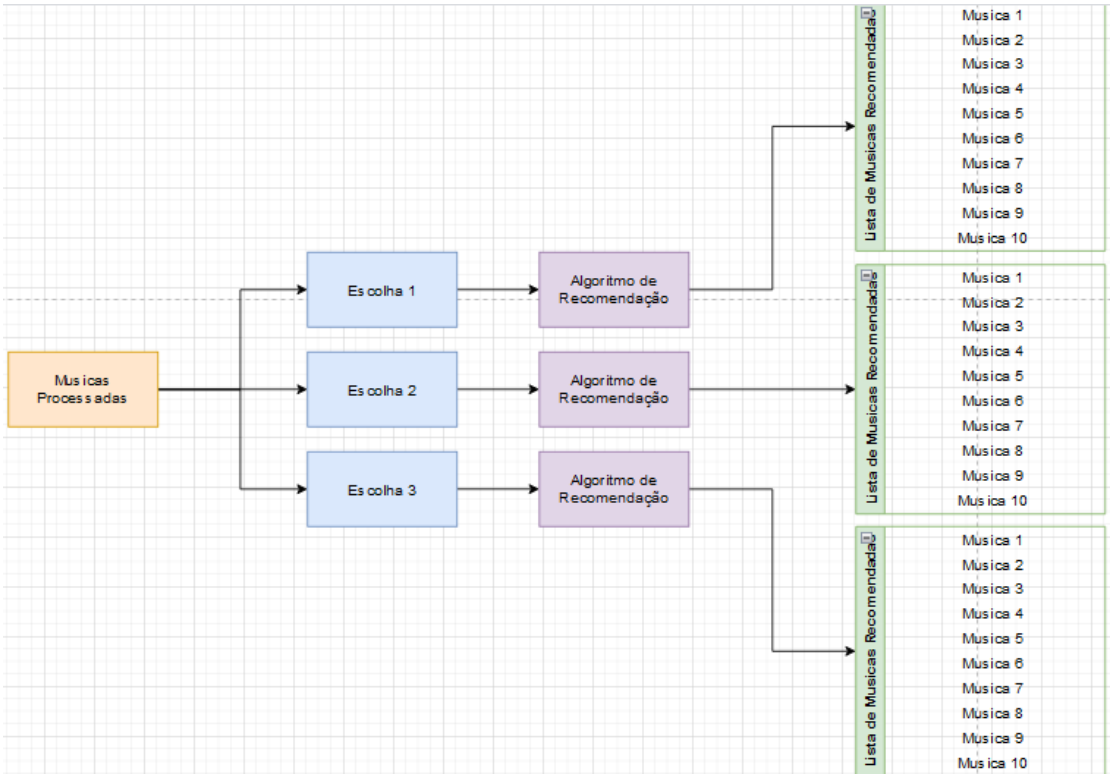

Figura 8 – Fluxograma do algoritmo de randomização de escolhas e geração de lista de recomendações. Fonte: Autoria própria.

5 Resultados

O método de recomendação de músicas foi desenvolvido para fornecer ao usuário uma seleção de dez faixas que são semelhantes à sua escolha inicial, conforme mencionado anteriormente. Para garantir a precisão desse sistema, dez músicas foram escolhidas aleatoriamente de todas as 3500 possíveis e essas recomendações foram processadas para cada escolha feita. Isso permite avaliar a similaridade entre as preferências do usuário e as amostras recomendadas.

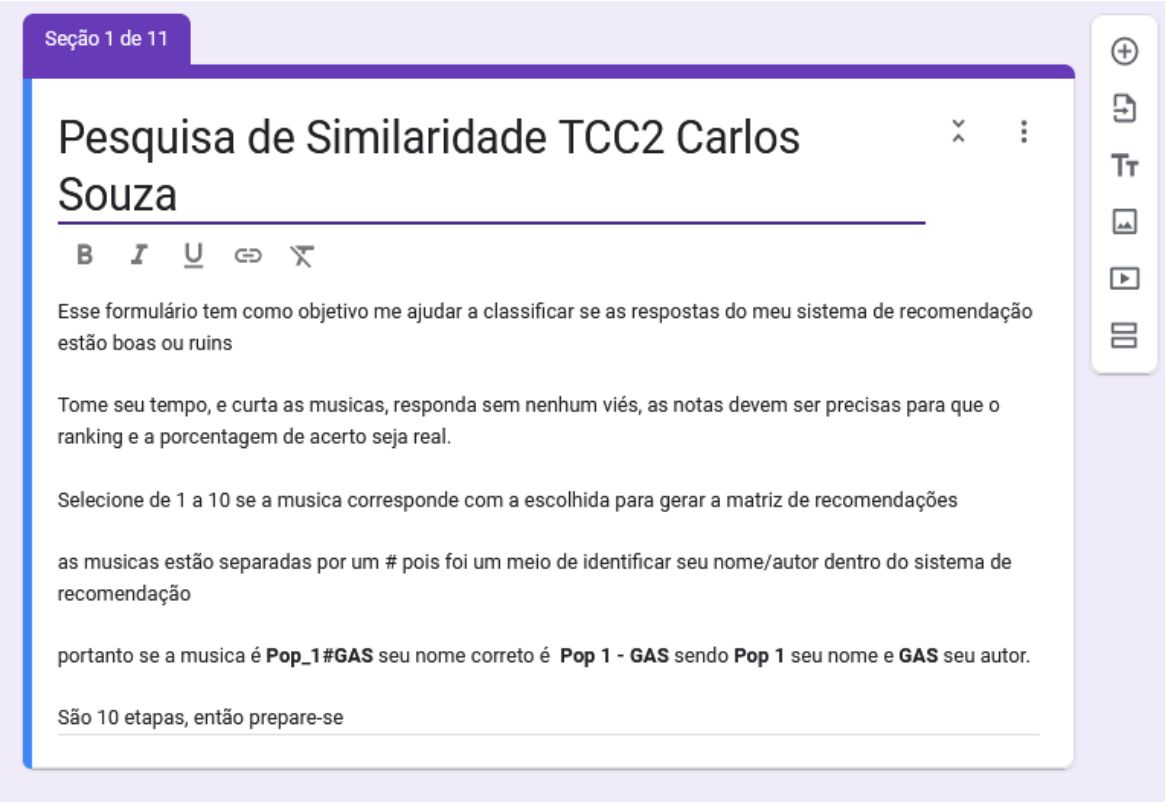

Figura 9 – Formulário disponibilizado para os voluntários realizarem suas análises e inserirem suas notas. Fonte: Autoria Própria.

A análise foi realizada por meio de um formulário conforme a Figura 9, disponibilizado para voluntários que se comprometeram a avaliá-las de forma imparcial. Uma cópia deste formulário de forma impressa estará presente ao final deste documento, denominado APÊNDICE A.

Os participantes dedicaram seu tempo para revisar e comparar as faixas sugeridas com as originais durante várias rodadas de análise. Desta forma foi possível calcular a média de acertos nas recomendações usando essas avaliações. Com isto

foi possível mensurar a eficácia do algoritmo de recomendação e a sua capacidade de atender às preferências musicais dos usuários. Conforme apresenta a Tabela 1.

Genero	Glitch	Rap	Orchestra	Phonk	Mpb	Cyberpunk	Trap	Forro	Glitch	Ambient
Media Nota 0-10	3,186	1,776	6,514	5,714	3,600	5,397	4,018	3,367	4,661	5,414
% Acerto	31,86%	17,76%	65,14%	57,14%	36,00%	53,97%	40,18%	33,67%	46,61%	54,14%
Total Geral	43,65%									

Tabela 1 – Tabela de resultados da análise das recomendações. Fonte: Autoria Própria.

As taxas de acerto variaram bastante entre os diferentes estilos musicais. Isso destaca áreas de sucesso e espaços para melhoria.

Para gêneros com maior predominância de um ritmo simples e presença de sons graves, como os gêneros Phonk e Orchestra, as taxas de de acerto foram ligeiramente acima da média, alcançando 57,14% e 65,14%, respectivamente. Por outro lado, gêneros como Rap, Glitch e Forró, que possuem uma predominância maior de sons agudos e uma ênfase considerável em letra e melodia, apresentaram taxas de acerto muito mais baixas, com 17,76%, 31,86% e 33,67%, respectivamente.

No âmbito geral, a taxa de acerto foi de aproximadamente 43,65%, indicando que o desempenho do algoritmo como um todo foi abaixo da média, evidenciando a necessidade de maiores aprimoramentos para lidar melhor com a diversidade de características musicais presentes em diferentes gêneros.

6 Considerações Finais

Os resultados alcançados mostram os pontos fortes e os pontos fracos do algoritmo para recomendar músicas.

A criação de um modelo de dados para abranger um espectro mais amplo de informações e variáveis foi crucial para aprimorar significativamente as recomendações oferecidas.

O resultado do algoritmo de pré-processamento dos dados permite identificar os pontos-chave que devem ser aprimorados. Esses dados destacaram a necessidade de considerar estratégias para o processamento de estruturas musicais complexas, além da análise detalhada dos pontos-chave de cada melodia. É crucial adotar algoritmos que possam lidar eficazmente tanto com a estrutura rítmica quanto com a melodia e a voz, o que é fundamental para a evolução contínua do processo de recomendação de músicas.

Além disso, é de se reconhecer a importância de explorar métodos para ponderar também os resultados dos vetores de saída, visando otimizar os dados ao reduzir seu tamanho sem comprometer significativamente a eficácia do algoritmo de recomendação. Este é um passo imprescindível para melhorar o desempenho geral do sistema de recomendação, assegurando que ele seja não apenas eficiente, mas também capaz de se adaptar dinamicamente às necessidades e preferências dos usuários.

O algoritmo de recomendação reflete diretamente os dados recebidos durante o processamento. Portanto, pré-classificar o gênero musical e outros dados gerais das faixas para separar os vetores de recomendação com base nessas características gerais pode significativamente melhorar a similaridade de cosseno e, consequentemente, a qualidade geral das recomendações.

Ao categorizar inicialmente as faixas por gênero musical e outras informações relevantes, o sistema pode criar clusters mais coesos de dados. Isso permite que o algoritmo de recomendação seja mais preciso ao calcular a similaridade entre músicas similares dentro desses clusters. Além disso, ao separar os vetores de recomendação

com base em características específicas, como tempo, tonalidade, instrumentação e outros atributos musicais, o sistema pode oferecer recomendações mais personalizadas e relevantes para os usuários.

6.1 Trabalhos Futuros

Com base nos resultados obtidos, identificam-se várias áreas promissoras para futuras pesquisas e desenvolvimento. O objetivo é aprimorar a precisão e a eficácia do algoritmo de recomendação de músicas, superando as limitações atuais e proporcionando uma experiência de usuário mais satisfatória. Dentre as diversas opções de melhoria identificado, as que apresentariam um maior impacto em relação com o objetivo principal a ser alcançado seriam:

1. **Desenvolvimento de Novos Métodos de Ponderação**: Uma área promissora é a investigação de métodos alternativos de ponderação de intervalos de tempo na música. Em vez de utilizar a média simples dos valores, propõe-se o uso de técnicas que ponderem os intervalos com base em critérios como intensidade e duração. Isso pode melhorar a precisão da análise de similaridade ao capturar melhor as nuances musicais.
2. **Diversificação das Características Musicais**: Ampliar o conjunto de características musicais consideradas pelo algoritmo é outro ponto importante. Além das métricas tradicionais, incorporar aspectos como ritmo, harmonia e timbre pode fornecer uma análise mais completa e precisa das faixas, resultando em recomendações mais personalizadas.
3. **Implementação de Técnicas Avançadas de Processamento de Sinais**: Explorar técnicas mais avançadas de processamento de sinais de áudio e análise de componentes principais, pode permitir a captura de detalhes mais sutis nas músicas.
4. **Estudo de Impacto de Estruturas Musicais Complexas**: Realizar estudos específicos sobre o impacto de estruturas musicais complexas é outro ponto que pode ser analisado. Músicas com dinâmicas variáveis e predominância de tons agudos podem influenciar negativamente a média de valores. Desenvolver métodos que minimizem esses efeitos pode levar a um aumento significativo na taxa de acertos.
5. **Integração com Plataformas de Streaming**: Testar e validar o algoritmo em ambientes reais, como plataformas de streaming de música, é uma etapa fundamental. Isso permitirá avaliar seu desempenho em larga escala e em

condições de uso real, além de fornecer dados valiosos para refinamentos contínuos.

6. **Implementação de novos métodos de validação**: A validação dos resultados por meio de pesquisa utilizando voluntários mostrou que intrinsecamente a análise não é imparcial, desta forma a análise dos resultados da similaridade utilizando outras formas, em principal outras ferramentas ou algoritmos é importante para que os resultados sejam mais concisos e possam ser exaustivamente testados, trazendo assim uma análise mais objetiva.

7. **Utilização de algoritmos para remoção de vocais**: Pode ser uma alternativa para processar a melodia e voz, buscando somente pela estrutura e ritmo, a remoção desses elementos pode melhorar a eficiência para os gêneros dos quais predominam, além de permitir uma análise mais focada à poucos elementos músicas por vez.

Essas direções de pesquisa visam não apenas superar as limitações identificadas, mas também explorar novas possibilidades para a melhoria contínua do algoritmo de recomendação, garantindo uma experiência mais rica e precisa para os usuários.

7 REFERÊNCIAS

BAHULEYAN, Hareesh. Music genre classification using machine learning techniques. **arXiv preprint arXiv:1804.01149**, 2018.

DANTAS, Joseclécio Dutra; DA SILVA CRUZ, Sergio. Um olhar físico sobre a teoria musical. **Caderno Brasileiro de Ensino de Física**, v. 41, n. 1, 2019.

ELBIR, Ahmet; AYDIN, Nizamettin. Music genre classification and music recommendation by using deep learning. **Electronics Letters**, v. 56, n. 12, p. 627-629, 2020.

Every Noise. [s.d.]. Every Noise at Once. Acesso em: [acesso em 12 maio 2024] . Disponível em: https://everynoise.com/

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. Editora Atlas SA, 2002.

GUPTA, Shikha et al. Feature extraction using MFCC. **Signal & Image Processing: An International Journal**, v. 4, n. 4, p. 101-108, 2013.

NDOU, Ndiatenda; AJOODHA, Ritesh; JADHAV, Ashwini. Music genre classification: A review of deep-learning and traditional machine-learning approaches. In: **2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)**. IEEE, 2021. p. 1-6.

PELCHAT, Nikki; GELOWITZ, Craig M. Neural network music genre classification. **Canadian Journal of Electrical and Computer Engineering**, v. 43, n. 3, p. 170-173, 2020.

Python Software Foundation. Python 3.9.7 Documentation [Internet]. 2021 [acesso em 12 maio 2024]. Disponível em: https://docs.python.org/3.9/.

SILLA, Carlos N.; KOERICH, Alessandro L.; KAESTNER, Celso AA. A machine learning approach to automatic music genre classification. **Journal of the Brazilian Computer Society**, v. 14, p. 7-18, 2008.

WAZLAWICK, Raul Sidnei. **Metodologia de pesquisa para ciência da computação**. Rio de Janeiro: Elsevier, 2014.

XIA, Peipei; ZHANG, Li; LI, Fanzhang. Learning similarity with cosine similarity ensemble. **Information sciences**, v. 307, p. 39-52, 2015.

XU, Changsheng; MADDAGE, Namunu Chinthaka; SHAO, Xi. Automatic music classification and summarization. **IEEE transactions on speech and audio processing**, v. 13, n. 3, p. 441-450, 2005.

APÊNDICE A – Formulário de pesquisa de similaridade

Pesquisa de Similaridade TCC2 Carlos Souza

Esse formulário tem como objetivo ajudar a classificar se as respostas do sistema de recomendação estão boas ou ruins

Tome seu tempo, e curta as músicas, responda sem nenhum viés, as notas devem ser precisas para que o ranking e a porcentagem de acerto seja real.

Selecione de 1 a 10 se a música corresponde com a escolhida para gerar a matriz de recomendações

as músicas estão separadas por um # pois foi um meio de identificar seu nome/autor dentro do sistema de recomendação

portanto se a música é **Pop_1#GAS** seu nome correto é **Pop 1 - GAS** sendo **Pop 1** seu nome e **GAS** seu autor.

Música Escolhida: Period_03#Sutekh

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
Both_Sides_Now#Judy_Collins										
Breathe#Télépopmusik_Angela_McCluskey										
Fade_Into_You#J_Mascis										
Haiti#Pan_Sonic										
Helplessly_Hoping_-_2005_Remaster#Crosby,_Stills_&_Nash										
I._F._A.#Phonophani										
Pop_1#GAS										
Pretty_Little_Fears_(feat._J._Cole)#6LACK_J._Cole										
Sanft_verblassen_die_Geschichten#Bersarin_Quartett										
Us_Against_the_World#Kanaya										

Música Escolhida: Collard_Greens#ScHoolboy_Q_Kendrick_Lamar

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
La-La_Means_I_Love_You#The_Delfonics										
Meta_Concrete#Rene_Hell										
Moanin'#Art_Blakey_&_The_Jazz_Messengers										
No_Children#The_Mountain_Goats										
Take_Me_Home,_Country_Roads#John_Denver										
The_Bug_Song#Chris_Cutler										
The_Recluse#Cursive										

Musica
Escolhida: Beethoven__Symphony_No._6_in_F_Major,_Op._68__Pastoral___I._Erwachen_heiterer_Empfindungen_bei_der_Ankunft_auf_dem_Lande._Allegro_ma_non_troppo#Ludwig_van_Beethoven_Philadelphia_Orchestra_Riccardo_Muti

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
5_Military_Marches,_Op._39,__Pomp_and_Circumstance___No._1_in_D_Major#Edward_Elgar_New_Zealand_Symphony_Orchestra_James_Judd										
Behind_The_Garage#Eric's_Trip										
Elgar__Variations_on_an_Original_Theme,_Op._36__Enigma___Variation_IX._Nimrod#Edward_Elgar_London_Symphony_Orchestra_Sir_Adrian_Boult										
Night_on_Bare_Mountain#Modest_Mussorgsky_Mariinsky_Orchestra_Valery_Gergiev										
'Round_Midnight#Thelonious_Monk										
Symphony_No._2_in_D_Major,_Op._73__I._Allegro_non_troppo#Johannes_Brahms_Philharmonia_Hungarica_Libor_Pešek										
Symphony_No._3_in_E_Flat_Major,_Op._97__Rhenish___II._Scherzo._Sehr_mäßig#Robert_Schumann_Staatskapelle_Berlin_Daniel_Barenboim										
Symphony_No._5_in_C_Minor,_Op._67__I._Allegro_con_brio#Ludwig_van_Beethoven_Wiener_Philharmoniker_Carlos_Kleiber										
Symphony_No._6_in_B_Minor,_Op._74,_TH_30__Pathétique___IV._Finale._Adagio_lamentoso_-_Andante#Pyotr_Ilyich_Tchaikovsky_Czech_Philharmonic_Semyon_Bychkov										

Symphony_No._8_in_C_Minor,_WAB_108_(Novak_1890_Version)__II._Scherzo._Allegro_moderato_-_Trio._Langsam#Anton_Bruckner_Tonhalle-Orchester_Zürich_Paavo_Järvi										

Música Escolhida: MIDNIGHT_RUN#GUCCIGARETTE_Tenngage

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
CORNER_STORE#Kilfiger										
Dead_Gods#Shinigami_Tenshi										
GET_SMACKED_IN_THIS_94'#Berrymane_Dj_Smokey_BACKWOOD_BOY										
I_Ain't_Trippin'#kushface_🕯										
Leanin'_Sidewayz#Robscire										
Plastic_Star#Byetone										
Simply_Put.#chromonicci										
Tel_Aviv_-_Recondite_Remix#Fauna_Flash_Recondite										
The_Sequences#Go_Yama										
Trapping_Stupid_Hard#Hymbeats										

Música Escolhida: Sujeito_De_Sorte#Belchior

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
Blues_Ain't_Nothin'#Johnny_Copeland										
Cheia_de_Charme#Guilherme_Arantes										
Do_You_Take_This_Man_#Diamanda_Galás										
Fom_Fonrom_Fonfom#Calango_Aceso										
Hot_Stuff#Donna_Summer										
People_Are_Strange#The_Doors										
Stuck_Like_Glue#Sugarland										
The_Best_Of_Me#The_Starting_Line										
Two_Dozen_Roses#Shenandoah										
Watching_You#Slave										

Música Escolhida: EDDIE#Still_Healing

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
Armz#Grantz										
INVU#TAEYEON										
Kiss_Me_Thru_The_Phone#Soulja_Boy_Sammie										
Mercury#HI-LO_Space_92_Oliver_Heldens										
Miss_The_Rage_(feat._Playboi_Carti)#Trippie_Redd_Playboi_Carti										
REVIVED#Derivakat										
Running_Through_My_Mind#DeJ_Loaf										
Swagon#Detlef_Ossey_James										
Toska#Tyrant [Free DL]										

Música Escolhida: Oh_U_Went_(feat._Drake)#Young_Thug_Drake

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
Cabra_Desmantelado#Sirano										
Cocaine_in_My_Brain#Dillinger										
Flex_(Ooh,_Ooh,_Ooh)#Rich_Homie_Quan										
From_The_D_To_The_A_(feat._Lil_Yachty)#Tee_Grizzley_Lil_Yachty										
I_Will_Return#Warm_Ghost										
Ms._Fat_Booty#Mos_Def										
MY_BIH_FOREIGN#Freddy_Konfeddy										
Tô_Bebendo,_Tô_Pagando#Sirano_Banda_Só_O_Mie										

Música Escolhida: Coração_Magoado#Solteirões_do_Forró

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
(Your_Love_Keeps_Lifting_Me)_Higher_&_Higher#Jackie_Wilson										
A_Fórmula_do_Amor_(feat._Kid_Abelha)_-_Remasterizado_2013#Leo_Jaime_Kid_Abelha										
A_Night_to_Remember_-_Single_Version#Shalamar										
Don't_Stop_The_Music#Yarbrough_&_Peoples										
Gashina#SUNMI										
Morango_Do_Nordeste#Frank_Aguiar										
Surrender#Cheap_Trick										

Música Escolhida: A16_-_The_Mole_MMD_Remax#Ultra-Red_The_Mole

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
Fitness_Force_Omega#Gridscape_Frank_B										
Mal_Sei_Falar_de_Amor#Raffé_Mainstreet										
PTSD#G_Herbo_Lil_Uzi_Vert_Juice_WRLD_Chance_the_Rapper										
Roly-Poly#T-ARA										
Shine#PENTAGON										
Tear_Drop#SF9										
Tiimmy_Turner#Desiigner										
You_Are_Mine#VICTON										

Música Escolhida: Two_Chambers#Loscil

Nome	Similaridade									
	1	2	3	4	5	6	7	8	9	10
A_Sea_Of_Love#Huerco_S.										
Autioitu_1#Ilpo_Väisänen										
I_Can't_Quit_You_Baby#Willie_Dixon										
Introspection_-_Edit#Laraaji										
Lunar_Eclipse#Alpha_Wave_Movement										
Minuetto#Luigi_Boccherini_Berliner_Symphoniker_Eduardo_Marturet										
Planner's_Beauty#JAB										
Stratum#Visible_Cloaks_Yoshio_Ojima_Satsuki_Shibano										
Symphony_No._5_in_C_Minor,_Op._67__I._Allegro_con_brio#Ludwig_van_Beethoven_Wiener_Philharmoniker_Carlos_Kleiber										
The_Seduction_of_Dr_Pasteur#Richard_Bone										

APÊNDICE B – Código Fonte

Algoritmo para renomear arquivos

```
import os
import eyed3
import re
import time
import concurrent.futures
import time

############################################
eyed3.log.setLevel("ERROR")
############################################

def time_format(time):
    """
    Formata o tempo em uma forma facil de ler.
    Args:
        time: valor em segundos de tempo.

    Returns:
        String formatada de tempo.
    """
    horas, rem = divmod(time, 3600)
    minutos, segundos = divmod(rem, 60)

    tempo = f"{int(horas)}:{int(minutos)}:{int(segundos)}"

    return tempo

def limpar_nome(nome):
    # Substitui caracteres inválidos e espaços por "_", e elimina caracteres '#'
    nome_sem_espacos_e_hashtags = re.sub(r'[\/:*?"<>| #]', '_', nome)
    return nome_sem_espacos_e_hashtags

def renomear_arquivos_mp3(pasta_origem):
    print(f"Renomeando arquivos MP3 em {pasta_origem}...")
    for arquivo in os.listdir(pasta_origem):
        if arquivo.endswith(".mp3"):
            caminho_arquivo = os.path.join(pasta_origem, arquivo)

            try:
                # Lê os metadados do arquivo MP3
                audiofile = eyed3.load(caminho_arquivo)

                # Obtém o nome da música e o nome do autor dos metadados
                nome_musica = limpar_nome(audiofile.tag.title)
                nome_autor = limpar_nome(audiofile.tag.artist)

                # Se ambos os metadados existirem, renomeia o arquivo
                if nome_musica and nome_autor:
                    novo_nome = f"{nome_musica}#{nome_autor}.mp3"
                    novo_caminho = os.path.join(pasta_origem, novo_nome)
                    os.rename(caminho_arquivo, novo_caminho)
                    print(f"Arquivo renomeado: {caminho_arquivo} -> {novo_caminho}")
            except Exception as e:
                print(f"Erro ao processar {caminho_arquivo}: {e}")


def pr_nomes(paths: list[str]) -> None:
    """
    Renomeia arqivos MP3 baseado em seus metadados usando processamento paralelo.

    Args:
        paths: Uma lista de paths para os arquivos MP3 a serem renomeados.

    Returns:
        None
    """
```

```
    inicio = time.time()

    with concurrent.futures.ThreadPoolExecutor() as executor:
        executor.map(renomear_arquivos_mp3, paths)

    fim = time.time()
    tempo = inicio - fim
    print(f"Tempo gasto: {time_format(tempo)} seconds")

if __name__ == "__main__":

    path = "/content/drive/My Drive/TCC/musics"

    map_genre = [os.path.join(path, genre) for genre in os.listdir(path) if
os.path.isdir(os.path.join(path, genre))]
    list_genre = os.listdir(path)

    #utiliza Thread para maximizar o I/O
    pr_nomes(map_genre)
```

Algoritmo para Preprocessamento dos vetores

```
import numpy as np
import librosa
import h5py
from pydub import AudioSegment
import random
import multiprocessing as mp
import os
import time

class AudioDataProcess:

    def __init__(self, audio_file):

        self.audio_file = audio_file
        audio = AudioSegment.from_file(audio_file)

        self.sample_rate = audio.frame_rate
        self.audio_signal = np.array(audio.get_array_of_samples()).astype(np.float32)


    def chromaStft(self):
        return librosa.feature.chroma_stft(y=self.audio_signal, sr=self.sample_rate)

    def chromaCqt(self):
        return librosa.feature.chroma_cqt(y=self.audio_signal, sr=self.sample_rate)

    def chromaCens(self):
        return librosa.feature.chroma_cens(y=self.audio_signal, sr=self.sample_rate)

    def chromaVqt(self):
        return librosa.feature.chroma_vqt(y=self.audio_signal, sr=self.sample_rate,
intervals='equal')

    def mfcc(self):
        return librosa.feature.mfcc(y=self.audio_signal, sr=self.sample_rate, n_mfcc=20)

    def spectrogram(self):
        return librosa.feature.melspectrogram(y=self.audio_signal, sr=self.sample_rate)

    def spectralFlatness(self):
        return librosa.feature.spectral_flatness(y=self.audio_signal)

    def spectralBandwidth(self):
        return librosa.feature.spectral_bandwidth(y=self.audio_signal, sr=self.sample_rate)

    def spectralCentroid(self):
```

```
        return librosa.feature.spectral_centroid(y=self.audio_signal, sr=self.sample_rate)

    def spectralRolloff(self):
        return librosa.feature.spectral_rolloff(y=self.audio_signal, sr=self.sample_rate)

    def spectralContrast(self):
        return librosa.feature.spectral_contrast(y=self.audio_signal, sr=self.sample_rate)

    def rms(self):
        return librosa.feature.rms(y=self.audio_signal)

    def polyFeatures(self):
        return librosa.feature.poly_features(y=self.audio_signal, sr=self.sample_rate)

    def tonnetz(self):
        return librosa.feature.tonnetz(y=self.audio_signal, sr=self.sample_rate)

    def zcr(self):
        return librosa.feature.zero_crossing_rate(y=self.audio_signal)

def time_format(time):
    """
    Formats a given time value in seconds into a human-readable string.
    Args:
        time: The time value in seconds.

    Returns:
        A formatted string representing the time.
    """
    hours, remainder = divmod(time, 3600)
    minutes, seconds = divmod(remainder, 60)

    elapsed_time_str = f"{int(hours)}:{int(minutes)}:{int(seconds)}"

    return elapsed_time_str

def save_ftv(ftv, format="npy"):
    """
    Saves the feature vector with a random name in the temp_result directory.

    Args:
        ftv: The feature vector to save.
        format: The desired format ("npy" or "h5"). Defaults to "npy".
    """

    os.makedirs("/content/drive/My Drive/TCC/vector_mean", exist_ok=True)

    random_name = "".join(random.choice("0123456789abcdefghijklmnopqrstuvwxyz") for _ in
range(60))

    if format == "npy":
        save_path = os.path.join("/content/drive/My Drive/TCC/vector_mean",
f"{random_name}.npy")
        np.save(save_path, ftv)
        print(f"Saved feature vector to {save_path}")
    elif format == "h5":
        save_path = os.path.join("/content/drive/My Drive/TCC/vector_mean",
f"{random_name}.h5")
        with h5py.File(save_path, "w") as f:
            f.create_dataset("ftv", data=ftv)
        print(f"Saved feature vector to {save_path}")
    else:
        raise ValueError(f"Invalid format specified: {format}")

def pr_musica(path,genre,music):
    """
    Processa uma música e retorna um vetor de características.

    Args:
        path: Caminho para o diretório que contém as músicas.
        genre: Gênero da música.
```

```
        music: Nome da música.

    Returns:
        Vetor de características da música.
    """
    start_time = time.time()

    path_music = os.path.join(path,genre, music)
    try:
        music_data = AudioDataProcess(path_music)
    except:
        print(f"Erro ao processar {music}")
        return 1

    chroma_stft = music_data.chromaStft()
    mfccs = music_data.mfcc()
    spectral_flatness = music_data.spectralFlatness()
    spectral_bandwidth = music_data.spectralBandwidth()
    spectral_centroid = music_data.spectralCentroid()
    spectral_rolloff = music_data.spectralRolloff()
    spectral_contrast = music_data.spectralContrast()
    rms = music_data.rms()
    poly_features = music_data.polyFeatures()
    zcr = music_data.zcr()

    feature_vector =
np.concatenate((mfccs,chroma_stft,spectral_contrast,spectral_flatness,spectral_bandwidth,spect
ral_centroid,spectral_rolloff,rms,poly_features,zcr), axis=0)
    feature_vector = feature_vector.T

    linhas_a_remover = np.where(feature_vector[:, 1] == 0)[0]
    feature_vector = np.delete(feature_vector, linhas_a_remover, axis=0)

    feature_vector = np.mean(feature_vector,axis=0)

    nomes = np.array([genre, music])
    ftv = np.append(nomes, feature_vector)

    save_ftv(ftv)

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"[T]: {time_format(elapsed_time)} s - {music} - {genre} - {ftv.shape}")

    return 0


def pre_prc(path, list):
    """
    Processa uma lista de diretórios

    Args:
        path: Caminho para o diretório que contém as músicas.
        list: lista de diretorios gênero da música.
    """

    start_time = time.time()
    try:
        os.rmdir("/content/drive/My Drive/TCC/vector_mean")
    except:
        pass

    with mp.Pool(processes=POOL) as pool:

        results = []

        for genre in list:
            dir = os.path.join(path, genre)
            files = os.listdir(dir)
            musics = [file for file in files if file.endswith(".mp3")]
```

```
            for music in musics:
                results.append(pool.apply_async(pr_musica, (path, genre, music)))

        for result in results:
            result.get()


    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Tempo gasto: {time_format(elapsed_time)} seconds")

#constantes
POOL = mp.cpu_count()

if __name__ == "__main__":

    path = "/content/drive/My Drive/TCC/musics"

    list_genre = os.listdir(path)

    print("CPUs utilizadas "+str(POOL))

    pre_prc(path, list_genre)

    exit();
```

Algoritmo para geração de recomendação

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import os

def load_data(file_paths):
    data = []
    for file_path in file_paths:
        array = np.load('/content/drive/My Drive/TCC/vector_mean/'+file_path)
        data.append(array)
    return np.vstack(data)

def calculate_similarity(data):
    similarity_matrix = cosine_similarity(data)
    return similarity_matrix

def generate_recommendations(similarity_matrix, song_index, num_recommendations):
    song_similarity = similarity_matrix[song_index]

    sorted_indices = np.argsort(song_similarity)[::-1]

    recommended_indices = sorted_indices[1:num_recommendations+1]

    return recommended_indices

if __name__ == "__main__":
    path = "/content/drive/My Drive/TCC/vector_mean"

    file_paths = os.listdir(path);

    data = load_data(file_paths)

    print(data.shape)
    file_names = data[:, 1]

    data = data[:, 2:].astype(float)

    similarity_matrix = calculate_similarity(data)
```

```
    for i in range(10):
        song_index = np.random.randint(0, data.shape[0])
        print('')
        print('Musica escolhida: ' + file_names[song_index])
        num_recommendations = 10
        recommendations = generate_recommendations(similarity_matrix, song_index,
num_recommendations)

        recommended_file_name = file_names[recommendations]
        print('Recomendações: ')
        print('')
        for musica in recommended_file_name:
          print(musica)

    exit();
```