Yoram Kraiser Goldstein

Sistema de recomendação de jogos digitais

São Paulo, SP

2024

Yoram Kraiser Goldstein

Sistema de recomendação de jogos digitais

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Ricardo Nakamura

São Paulo, SP

2024

Catalogação-na-publicação

Goldstein, Yoram
Sistema de recomendação de jogos digitais / Y. Goldstein -- São Paulo, 2024.
63 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Sistemas de Recomendação 2.Jogos Digitais I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Este trabalho é dedicado às nossas queridas famílias e amigos, cujo apoio incondicional e amor são fontes inestimáveis de força e inspiração.

Agradecimentos

Gostaria de expressar meus sinceros agradecimentos a todos que contribuíram para o desenvolvimento e conclusão deste projeto. Agradeço ao Prof. Dr. Ricardo Nakamura pelo acompanhamento como orientador do projeto durante o desenvolvimento do projeto. Agradeço também à minha familia e aos meus amigos pelo apoio durante a graduação.

Resumo

Este trabalho tem como objetivo explorar o universo dos sistemas de recomendação, com foco principal nas estratégias utilizadas por desenvolvedores de programas de recomendação. Como a indústria de jogos digitais está em constante crescimento, os jogadores têm acesso a uma quantidade enorme de opções, o que pode ser ao mesmo tempo empolgante mas também confuso. Sistemas de recomendação ajudam a filtrar essa grande quantidade de informações, tornando mais fácil para os jogadores encontrarem títulos que realmente apreciem. Com isso, pode-se ajudar novos e experientes jogadores a encontrar jogos que combinem com seus gostos e preferências. Como resultado, foi desenvolvido um protótipo de sistema de recomendação que faz a recomendação de jogos com base em informações inseridas pelo usuário.

Palavras-chave: Sistemas de Recomendação. Jogos de Vídeo. C++. CGICC.

Abstract

This work has as its objective to explore the universe of recommendation systems, with its main focus on the strategies used by recommendation program developers. As the video games industry is in constant growth, the players have access to a huge amount of options, which can be both exciting and disorienting. Recommendation systems help filter this great quantity of information, making it easier for players to find titles that they truly appreciate. With that, it is possible to help new and returning players to find games that fit their likes and preferences. As a result, we developed a prototype of a recommendation system that recommends games based on information inserted by the user.

Keywords: Recommendation Systems. Video Games. C++. CGICC.

Lista de ilustrações

Sumário

1 Introdução

Sistemas de recomendação são softwares que preveem automaticamente o quanto um usuário irá gostar de um determinado item (ANWAR et al., 2017). Estes sistemas se utilizam de diversas estratégias e sugerem opções, ou escolhas dentro de um determinado tema e relevantes para um determinado usuário ou grupo de usuários. O Spotify é um exemplo de sistema de recomendação que utiliza algoritmos para analisar as preferências musicais dos usuários, histórico de escuta e hábitos de navegação para recomendar músicas, artistas e playlists que provavelmente serão do interesse do usuário. Outros exemplos são as plataformas de comércio digital, serviços como redes sociais e notícias, além de plataformas de venda e distribuição de jogos como o Steam.

Segundo Cheuque, Guzmán e Parra (2019) sistemas de recomendação de jogos, são sistemas capazes de fazer sugestões personalizadas relevantes, alertando os usuários sobre jogos desconhecidos assim como novos lançamentos. Assim como a indústria musical, a indústria de jogos se voltou para uma estratégia de vendas digitais, baseadas em plataformas online. Isso traz para o usuário uma vasta disponibilidade e possibilidade de acessos a novos jogos. Diante dessa vasta gama de escolha, os sistemas de recomendação são cada mais necessários para guiar e auxiliar usuários na busca por jogos coerentes com seus gostos e novidades pertinentes.

A plataforma Steam (Valve Corporation,) assim como o Spotify (Spotify AB,) busca preferências do usuário (CHEUQUE; GUZMáN; PARRA, 2019) coletando dados como histórico de compras, ou avaliações dos jogos, entre outros fatores para propor opções de novas compras, podendo ser jogos, novos ou mesmo jogos antigos que nunca foram jogados pelo usuário.

Este projeto tem como objetivo desenvolver um sistema de recomendações e experimentar estratégias e algoritmos capazes de guiar novos usuários, de plataformas de jogos online, na escolha de jogos baseados em preferências e dados do perfil do usuário. O sistema proposto vai filtrar e cruzar informações dos jogos como: data de lançamento, gênero e empresa desenvolvedora e informações propostas pelo usuário como tipos de jogos que o usuário possui, gêneros que prefere entre outras informações. Para se validar o objetivo, será implementado um protótipo do sistema, que será avaliado por usuários por critérios qualitativos sobre as recomendações produzidas.

1.1 Justificativa

A indústria de jogos digitais está em constante crescimento, com novos lançamentos acontecendo regularmente (LIMA, 2024; SteamDB, 2023). Os jogadores têm acesso a uma quantidade enorme de opções, o que pode ser ao mesmo tempo empolgante mas também confuso. O conceito da economia de atenção (BENEDITO, 2024) acaba sendo um fator de importância para o consumidor, pois com cada vez mais jogos diferentes tentando atrair sua atenção, seria de grande benefício poder focar-se apenas nos que mais os interessa. Sistemas de recomendação ajudam a filtrar essa grande quantidade de informações, tornando mais fácil para os jogadores encontrarem títulos que realmente apreciem. A criação de um sistema de recomendação de jogos digitais atende a duas necessidades principais. Primeiro, pode-se ajudar novos jogadores a encontrar jogos que combinem com seus gostos e preferências.

Além disso, este projeto visa um aprofundamento no estudo dos algoritmos e técnicas utilizadas em sistemas de recomendação. Durante o desenvolvimento do protótipo, vão-se aprofundar em diferentes métodos, como a filtragem colaborativa e a filtragem baseada em conteúdo. Este processo permitirá um aprendizado prático sobre como esses algoritmos funcionam, suas vantagens e limitações, e como podem ser aplicados em diferentes contextos.

1.2 Objetivos

O objetivo principal deste projeto de TCC é explorar o universo dos sistemas de recomendação, com foco principal nas estratégias utilizadas por desenvolvedores de programas de recomendação. A partir desse estudo, pretende-se desenvolver um protótipo de sistema de recomendação de jogos utilizando uma das estratégias analisadas. Os objetivos específicos incluem a experimentação de diferentes algoritmos e técnicas de recomendação, bem como a aplicação de estratégias de desenvolvimento de projetos em C++. A validação do protótipo será feita de forma qualitativa a partir de estudos com usuários.

2 Sistemas de Recomendação

Os sistemas de recomendação são importantes na era digital porque ajudam as pessoas a encontrar o que procuram em meio a uma grande quantidade de informações. Eles tornam a navegação mais fácil e melhoram a experiência do usuário em várias plataformas, como e-commerce, serviços de streaming e redes sociais . Ao personalizar sugestões com base nas preferências e comportamentos dos usuários, esses sistemas aumentam a satisfação e o engajamento dos usuários (CAKIR; ARAS, 2012; LOPS; GEMMIS; SEMERARO, 2011; Lü et al., 2012). No caso dos jogos digitais, os sistemas de recomendação são essenciais para ajudar os jogadores a descobrir novos títulos que combinam com seus gostos e histórico de jogo, facilitando a exploração de novos conteúdos. Este trabalho apresentará algumas estratégias básicas e algoritmos usados nesses sistemas.

Na exploração do tema em pesquisa bibliográfica, levantou-se algumas abordagens principais para a personalização de buscas e sistemas de recomendação. São elas: sistemas baseados em filtragem de conteúdo (LOPS; GEMMIS; SEMERARO, 2011; THORAT; GOUDAR; BARVE, 2015), filtragem colaborativa (ANWAR et al., 2017; Lü et al., 2012) e regras de associação (CAKIR; ARAS, 2012; KIM; KIM, 2003). Na filtragem baseada em conteúdo, as descrições de conteúdo dos itens são representadas por um perfil de usuário, e itens semelhantes ao perfil do usuário são recomendados para ele. A filtragem colaborativa, se baseia nas semelhanças entre usuários em uma comunidade de usuários onde por exemplo as avaliações de um jogo por um usuário são comparadas com usuários semelhantes, gerando recomendações para jogos que o usuário ainda não avaliou ou nunca jogou. E finalmente os sistemas baseados em regras de associação, buscam padrões nas comparações entre perfis de usuários e Identifica itens que são frequentemente comprados ou utilizados juntos. Descrevemos também que é possível misturar essas estratégias criando sistemas híbridos.

2.1 Recomendação por filtragem e filtragem baseada em conteúdo

Em um sistema básico de recomendação a filtragem simples é uma primeira estratégia para reduzir o tamanho da lista de itens em uma grande base de dados. O usuário seleciona algumas características extraídas previamente dos jogos e a filtragem manual funciona como uma primeira fase de triagem. O usuário pode assim escolher filtrar categorias de jogos como ação ou aventura ou escolher eliminar jogos de consoles que ele não possui. Este filtro ajuda a reduzir o número de itens e apresenta ao usuário apenas os que correspondem a esses critérios básicos. Nos filtros básicos do nosso projeto inicial, o sistema permite que o usuário aplique critérios simples, como selecionar jogos com base em

certas características como data de lançamento (ex: jogos lançados depois de 2020 ou que estão disponíveis em uma plataforma específica como PlayStation) ou mesmo a censura idade. Com esse filtro uma família que deseja evitar jogos violentos pode excluir jogos com censura maior de 16 anos. Embora úteis, esses filtros têm limitações, especialmente quando o objetivo é descobrir novos jogos ou obter recomendações automatizadas e relevantes.

A filtragem baseada em conteúdo por outro lado se baseia na criação prévia de um perfil de usuário que contém as preferências já explicitadas pelo usuário, mas também dados extraídos de seu uso do sistema. Essa estratégia se torna agora mais automatizada e enriquecida por mais informações indiretas do usuário. Assim, além dos dados fornecidos manualmente pelo usuário, o sistema agora conta com informações como a quantidade de tempo que o usuário jogou certos jogos ou histórico de compras ou desejos (wishlists) explicitados. Este perfil é então comparado às informações e características dos jogos. Este método não leva em conta as preferências de outros usuários, focando apenas nas preferências individuais.(LOPS; GEMMIS; SEMERARO, 2011)

2.1.1 Extração de Características

Para que essas duas estratégias de filtro funcionem é necessário inicialmente se representar esses jogos em uma base de dados e tornar possível filtrar ou comparar os elementos, no nosso caso, os jogos. A extração de características é o processo de identificar e selecionar atributos específicos de um item que serão utilizados para fazer as recomendações. No contexto de jogos, essas características podem incluir vários aspectos relevantes que descrevem um jogo. Essa extração pode ser manual, ou automatizada e pode ter vários níveis de complexidade como veremos para as outras estratégias como a filtragem colaborativa ou a associação de regras. No nosso caso inicial foi escolhida uma lista previamente criada por usuários do site https://www.kaggle.com/datasets. No nosso caso foi uma lista CSV criada por Matheus Fonseca Chaves que será descrita em detalhes no tópico banco 3.2. (CHAVES, 2023)

As características escolhidas pelo usuário são atributos ou descritores que ajudam a definir e diferenciar cada jogo de maneira coerente e relevante para todos os jogadores e mais a frente que elas serão necessárias para serem comparadas com perfis do usuário.

Exemplos de Características Relevantes para jogos:

- Tipo: O tipo de jogo, como ação, aventura, estratégia, RPG, etc.
- Plataforma: As plataformas onde o jogo está disponível, como PC, PlayStation, Xbox, Nintendo Switch, etc.
- Desenvolvedor: A empresa ou estúdio que desenvolveu o jogo.
- Data de Lançamento: Quando o jogo foi lançado.

- Avaliações: Pontuações ou classificações dadas pelos usuários.
- Gênero: O tema ou ambientação do jogo, como fantasia, ficção científica, histórico, etc.
- Modo de Jogo: Se o jogo é single-player, multiplayer, co-op, etc.
- Gráficos e Estilo Visual: O estilo visual do jogo, como 2D, 3D, pixel art, etc.

No nosso projeto as características dos jogos são representadas por meio de uma classe com strings e vetores de strings, sendo que uma string única representa uma característica singular do jogo (e.g. data de lançamento, modo de jogo) e um vetor de strings representa uma característica da qual podem existir múltiplas para um único jogo (e.g. desenvolvedores, plataformas, gêneros). Por outro lado criaremos também uma classe para o perfil de usuário que possuirá seus próprios strings e vetores de strings, representando quais características que o usuário procura. Em um momento seguinte, se fará a comparação das duas classes.

2.1.2 Perfil do Usuário

As informações do perfil do usuário são também uma forma de extração de características. De um lado o sistema pode buscar informações pessoais sobre compra de jogos, sobre quanto tempo o usuário jogou, um ou mais jogos, e outras informações pessoais geradas pelo usuário como localização, histórico de jogos e interações com a plataforma. Esses dados são usados para entender os gostos do usuário e fornecer recomendações mais precisas e personalizadas (MATOS, 2022). Por exemplo, se um usuário tem um histórico de jogar e gostar de jogos de estratégia, o sistema pode priorizar recomendações dentro desse gênero. mas que ele não expressamente inseriu no sistema (implícitas). Por outro lado, o usuário pode inserir dados pessoais como dados familiares, quantidade de filhos, endereço, etc.)(explícitas) (BARTH, 2010).

Para se extrair essas características numéricas pode se contar o número de jogos de um certo gênero que ele possui mas também pode-se levar em conta a quantidade de tempo que jogador dedicou a certos tipos de jogos

A criação de perfis de usuário envolve coletar e analisar essas informações, como os tipos de jogos que o usuário jogou, avaliações que ele deu a diferentes jogos, e padrões de comportamento que serão usados em seguida em diferentes níveis de complexidade ou de acordo com as estratégias apresentadas a seguir. Esse perfil permite ao sistema identificar similaridades e diferenças entre usuários também nas estratégias de filtragem-colaborativa e de maneira ainda mais automatizada relacionando padrões entre perfis de usuários e características de jogos nas estratégias de associação de regras apresentados nos capítulos mais a frente. Mas em geral a criação de perfil de usuário é uma área de pesquisa em si

que trata de diversos aspectos da mineração de dados, que trata também de questões de privacidade etc.. No nosso projeto nós vamos partir de um perfil criado manualmente e não vamos criar um mecanismo de mineração.

O que nos interessa é testar os algoritmos que permitem automatizar as comparações entre perfis de usuários e as características dos jogos e as formas de sugerir jogos coherentes dentro de uma grande lista de jogos. Para isso existem métodos para medir a similaridade entre perfis de usuários e identificar jogos coerentes com seus interesses, há vários métodos que podem ser aplicados. Entre eles, destacam-se as métricas de similaridade, como a distância Euclidiana e a correlação de Pearson, que comparam diretamente as características dos perfis.

2.1.3 Métodos de Similaridade

Para podermos comparar diferentes itens com um perfil de usuário, precisamos utilizar métodos de similaridade. Métodos de similaridade são utilizados para medir a correspondência entre objetos, como documentos, usuários ou itens, e desempenham um papel importante em diversas áreas, como sistemas de recomendação, processamento de linguagem natural e aprendizado de máquina. Esses métodos avaliam quão próximos ou relacionados estão dois conjuntos de dados, permitindo identificar itens semelhantes, agrupando objetos relacionados, e executando tarefas de classificação e predição. Assim, essas técnicas ajudam a automatizar o processo de encontrar padrões e recomendações baseados nas similaridades entre perfis e itens. Nesta seção serão apresentados diversos tipos diferentes de métodos de similaridade que são comumente utilizados.

2.1.3.1 Similaridade por Cosseno

Utilizada para medir o cosseno do ângulo entre dois vetores. É particularmente útil em sistemas de processamento de texto para comparar documentos representados como vetores de termos.

Exemplo no Protótipo

Vamos criar um exemplo básico que poderia ser implementado no nosso protótipo onde temos dois jogos e queremos calcular a similaridade entre eles com base em suas características. Suponha que cada jogo seja representado por um vetor de características. Para simplificar, consideramos três características: ação, aventura e estratégia. Suponha que temos dois jogos

Jogo 1: [ação = 3, aventura = 1, estratégia = 2]

Jogo 2: [ação = 2, aventura = 2, estratégia = 3]

Esses vetores representam a intensidade de cada característica para os jogos, em

uma escala de 0 a 10. E vejamos como podemos fazer o cálculo de similaridade de Cosseno:

1. Produto Escalar:

$$\vec{A} \cdot \vec{B} = \sum_{i=1}^{n} A_i * B_i$$

O produto escalar entre dois vetores A e B mede a soma dos produtos dos componentes correspondentes desses vetores. É um valor escalar que representa a magnitude da projeção de um vetor sobre o outro. Este valor é necessário porque o cosseno do ângulo entre dois vetores depende da relação entre os componentes correspondentes desses vetores. Como exemplo temos

$\vec{A} \cdot \vec{B}$ = (Ação jogo 1 × Ação jogo 2) + (Aventura jogo 1 × Aventura jogo 2) + (Estratégia jogo 1 × Estratégia jogo 2)

$\vec{A} \cdot \vec{B} = (3 * 2) + (1 * 2) + (2 * 3) = 6 + 2 + 6 = 14$

2. Norma dos Vetores:

$$||\vec{A}|| = \sqrt{\sum_{i=1}^{n} A_i^2}$$

A norma de um vetor é a raiz quadrada da soma dos quadrados de seus componentes. A norma de um vetor A é a distância do ponto final do vetor até a origem no espaço multidimensional. Para calcular a similaridade por cosseno, precisamos da norma para normalizar o produto escalar. Sem a normalização, o produto escalar sozinho não nos fornece uma medida de similaridade independente da magnitude dos vetores.

Jogo 1: [ação = 3, aventura = 1, estratégia = 2]

Para o vetor A = [3,1,2]: $||\vec{A}|| = \sqrt{3^2 + 1^2 + 2^2} = \sqrt{14}$

Jogo 2: [ação = 2, aventura = 2, estratégia = 3]

Para o vetor B = [2,2,3]: $||\vec{B}|| = \sqrt{2^2 + 2^2 + 3^2} = \sqrt{17}$

3. Similaridade por Cosseno: $S = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| * ||\vec{B}||} = \frac{14}{\sqrt{14} * \sqrt{17}} \approx 0.907$

A similaridade por cosseno é a razão entre o produto escalar dos vetores e o produto de suas normas. Esta relação nos dá o cosseno do ângulo entre os dois vetores. A razão pela qual se normaliza o produto escalar pelo produto das normas é para remover o efeito das magnitudes dos vetores, resultando em uma medida de similaridade que é independente do comprimento dos vetores. Isso significa que a similaridade por cosseno mede apenas a orientação relativa dos vetores, não sua magnitude absoluta. A similaridade por cosseno entre os dois jogos é aproximadamente 0.907, indicando uma alta similaridade entre eles.

2.1.3.2 Correlação de Pearson

Mede o grau de correlação linear entre dois conjuntos de dados, ajustando para diferentes mídias e escalas (AGGARWAL, 2016). É comumente usada para comparar preferências ou avaliações em sistemas de recomendação. No caso do nosso protótipo precisaríamos de um grande número de dados de usuários diferentes. Para este projeto estamos experimentando o uso de apenas um usuário para recomendar os jogos.

2.1.3.3 Distância Euclidiana

Calcula a distância geométrica direta entre dois pontos (ou vetores) em um espaço multidimensional. Nesse caso, a distância euclidiana interpreta a dissimilaridade diretamente como uma "distância"no espaço, útil para medir o quão longe dois pontos estão.

Em sistemas de recomendação, a distância euclidiana é frequentemente utilizada para medir a semelhança entre os usuários ou itens. O conceito é que cada usuário ou item pode ser representado como um ponto em um espaço multidimensional, onde cada dimensão representa uma característica ou uma avaliação de um item específico. Ao calcular a distância euclidiana entre esses pontos, é possível estimar quão similares são dois usuários ou itens.

Este algoritmo foi escolhido para um segundo protótipo do sistema, pois acreditou-se que as características do algoritmo se assimilaram às diversas dimensões dos dados usados. Comparando as características de jogos da base de dados com as mesmas de jogos dados pelo usuário que ele confirma que gosta, podemos definir a distância euclidiana entre eles. Quanto menor a distância, mais próximo um jogo do outro, e maior a chance do usuário gostar desse jogo também.

No projeto final, não foi usado a distância euclidiana.

2.1.3.4 Similaridade de Jaccard

A similaridade de Jaccard, também conhecida como coeficiente de Jaccard, é uma métrica usada para medir a semelhança e a diversidade entre conjuntos de amostras. (Han et al., 2012) O coeficiente compara membros de dois conjuntos para ver quais membros são compartilhados e quais são distintos. É útil em várias aplicações, incluindo ecologia, estudos de biodiversidade, e mais popularmente, em sistemas de recomendação, para comparar a similaridade entre usuários ou itens.

O projeto não utiliza a similaridade de Jaccard pois os itens que o sistema compara no projeto são itens singulares, e não conjuntos.

2.1.3.5 Similaridade de Jaccard

A Distância de Manhattan, também conhecida como distância de táxi ou norma L1, (BISHOP, 2006) é uma métrica usada para medir a distância entre dois pontos em um espaço de grade, como o layout de ruas de uma cidade. O nome "Manhattan"é derivado do famoso arranjo de grade das ruas de Manhattan, onde você só pode se mover ao longo das ruas virando em ângulos retos, em vez de seguir uma linha direta diagonal.

Ele calcula a soma das diferenças absolutas entre as coordenadas de dois pontos. É útil em ambientes onde se quer considerar apenas o deslocamento em linhas retas em grades, como em planejamento urbano ou em certos tipos de problemas de otimização.

A Distância de Manhattan é uma abordagem mais simples, que considera cada característica de forma individual. Esse método é ideal quando o foco é apenas medir o quanto dois itens diferem em cada característica separadamente. Ela trabalha como se fosse um sistema de pontuação incremental e usando uma metáfora gráfica, como se ela calculasse distância em uma cidade com suas ruas e esquinas, enquanto que a distância euclidiana calcula todos os atributos em uma mesmo cálculo proporcionalmente criando uma linha reta entre os diversos itens comparados. A distância de Manhattan é especialmente útil para trabalhar com atributos que não precisam ser analisados de maneira proporcional, como categorias ou classificações discretas.

Comparando com a Distância Euclidiana, que trata as características de forma conjunta e proporcional, a distância de Manhattan tem menos variações e sutilezas. Dessa forma, a Distância Euclidiana é mais adequada quando se quer avaliar a relação entre todas as características ao mesmo tempo, para entender como elas, juntas, influenciam a similaridade entre os itens.

2.1.3.6 Distancia de Hamming

A Distância de Hamming é uma métrica usada para medir a diferença entre dois strings de igual comprimento, contando o número mínimo de substituições necessárias para transformar um string no outro. (HEFEZ; VILLELA, 2008; MENEGHESSO, 2012)

Desenvolvida por Richard Hamming em 1950, essa métrica é especialmente útil em campos como teoria da informação e codificação de erro, onde é fundamental detectar e corrigir erros em strings de dados transmitidos.

A distância de Hamming é calculada identificando posições nas quais os dois strings correspondentes diferem. Por exemplo, a distância de Hamming entre "karat"e "karma"é 2, porque as alterações necessárias estão nas terceira e quinta posições (’r’ para ’m’ e ’t’ para ’a’). Essa métrica ajuda na detecção de erros em dados codificados e na verificação de integridade de dados.

A aplicação da distância de Hamming em sistemas de recomendação é menos direta

do que medidas como a distância euclidiana ou a similaridade de Jaccard, porque ela se baseia na comparação de vetores ou strings de igual comprimento. Contudo, ainda há usos potenciais:

1. Recomendações Baseadas em Características Binárias:

Se os itens ou usuários podem ser descritos por vetores de características binárias (por exemplo, a presença ou ausência de certas características), a distância de Hamming pode ser usada para medir a similaridade entre esses vetores. Itens ou usuários com menor distância de Hamming teriam características mais similares.

Filtragem de Conteúdo com Metadados Codificados:

Em casos onde metadados ou características podem ser expressos como strings ou vetores binários, a distância de Hamming pode ajudar a identificar itens com maior grau de similaridade de metadados.

No caso do projeto atual, concluímos que a distância de Hamming poderia ser útil para o projeto pois ela pode ser usada para medir a similaridade entre o perfil de um jogador e os jogos disponíveis. Por exemplo, um jogador pode ter preferências de gênero representadas por um vetor binário (como [1, 0, 1, 0] para Aventura e RPG), e os jogos também podem ser representados dessa forma. A distância seria calculada ao comparar as diferenças nas posições desses vetores.

Essa abordagem é útil quando se deseja identificar diferenças claras em características categóricas, como gêneros de jogos. Ao medir quantos atributos entre um jogador e um jogo diferem, a distância de Hamming pode ajudar a encontrar jogos que correspondem mais de perto às preferências do jogador. No entanto, ela não considera a intensidade das diferenças, focando apenas na presença ou ausência de divergências, o que pode ser uma limitação quando se deseja capturar nuances mais complexas nas características dos jogos.

2.2 Filtragem Colaborativa

Nas seções anteriores, discutimos a filtragem baseada em conteúdo, que utiliza as características dos itens e as preferências declaradas pelos usuários para fazer recomendações personalizadas. Essa abordagem foca diretamente nas descrições dos itens, sugerindo novos itens com base nas preferências já demonstradas por um usuário. Segundo Thorat, Goudar e Barve (2015), essa técnica garante um resultado mais fiel ao gosto do usuário pois conta com dados precisos e informações fornecidas pelo próprio usuário.

Agora, vamos explorar uma abordagem diferente, conhecida como filtragem colaborativa. Filtragem colaborativa (FILHO, 2010; MUREL; KAVLAKOGLU, 2024) é uma técnica amplamente utilizada em sistemas de recomendação para sugerir itens a usuários com base nas preferências de outros usuários semelhantes. A ideia principal por

trás da filtragem colaborativa é que, se um grupo de usuários compartilha preferências ou gostos similares, as escolhas feitas por uns podem ser úteis para recomendar itens a outros membros do grupo de interesse.

A principal diferença entre a filtragem colaborativa e a filtragem baseada em conteúdo está na fonte de informação utilizada para gerar recomendações. A filtragem baseada em conteúdo analisa as características dos itens (como gênero, tema ou estilo) e sugere novos itens semelhantes àqueles que o usuário já demonstrou interesse, com base em seu histórico de preferências. Já a filtragem colaborativa, em vez de focar nas características dos itens, faz recomendações com base nas preferências de outros usuários com perfis ou gostos semelhantes, assumindo que usuários que gostaram dos mesmos itens no passado continuarão a compartilhar interesses no futuro. Dessa forma, a filtragem colaborativa pode identificar novas descobertas, enquanto a filtragem baseada em conteúdo se concentra em itens com características similares (FILHO, 2010).

A Figura 1 mostra uma comparação de características entre as filtragens baseada em conteúdo e a colaborativa.

Baseados em conteúdo	***Filtragem Colaborativa***
Dependente de conteúdo	Independente de Conteúdo
Funciona com novos usuários	Não funciona com novos usuários
Não possui o problema da esparsidade	Problema da esparsidade
Não utiliza impressões humanas	Utiliza impressões humanas
Normalmente gera recomendações muito óbvias	Pode gerar recomendações surpreendentes

Figura 1 – Comparação entre tipos de sistema de filtragem (FILHO, 2010)

Vantagem

A filtragem colaborativa pode sugerir novos itens que o usuário-alvo pode não ter considerado anteriormente, mas que são relevantes e atraentes para seu perfil. (MUREL; KAVLAKOGLU, 2024)

Desvantagem

O "cold start"é uma desvantagem significativa nos sistemas de filtragem colaborativa, ocorrendo quando um novo usuário ou item é introduzido no sistema sem um histórico prévio de interações, dificultando a associação com usuários existentes. Sistemas baseados em conteúdo lidam melhor com novos itens, mas também enfrentam desafios com novos usuários.

Além disso, a escassez de dados em sistemas colaborativos é problemática, onde a falta de informações sobre as preferências dos usuários para a maioria dos itens leva à esparsidade de dados, tornando os modelos preditivos menos eficazes. Para combater isso, técnicas como fatoração de matrizes e decomposição em valores singulares são usadas para reduzir a dimensionalidade e melhorar as recomendações. Métodos que envolvem

usuários avaliando itens também são implementados para enriquecer os dados disponíveis. (MUREL; KAVLAKOGLU, 2024)

2.2.1 Filtragem Colaborativa Baseada em Usuários

A Filtragem Colaborativa Baseada em Usuários é uma técnica comum em sistemas de recomendação, que se baseia na premissa de que usuários que concordaram no passado tendem a concordar novamente no futuro. Este método é utilizado para prever as preferências de um usuário com base nas preferências de outros usuários com gostos semelhantes. Esta técnica se apoia em uma grande quantidade de usuários e que tem o hábito de classificar, no sentido de pontuar, avaliar os jogos.

Esses desafios têm levado ao desenvolvimento de métodos híbridos, que combinam a filtragem colaborativa com outras técnicas, como filtragem baseada em conteúdo, para melhorar o desempenho e a precisão dos sistemas de recomendação.

Alguns exemplos que temos na atualidade de sistemas de recomendação que utilizam esse algoritmo são:

- Netflix: Uma das aplicações mais famosas de filtragem colaborativa é no serviço de streaming Netflix, que recomenda filmes e séries de TV baseados nas avaliações e no comportamento de visualização de usuários com gostos semelhantes.
- Amazon: A Amazon utiliza filtragem colaborativa para recomendar produtos aos usuários com base no histórico de compras e nas avaliações de outros clientes que compraram e avaliaram produtos similares.
- Spotify: O serviço de streaming de música Spotify recomenda músicas e playlists analisando as preferências musicais dos usuários e comparando-as com as de outros usuários que têm gostos semelhantes.

2.2.2 Filtragem Colaborativa Baseada em Itens

Outra técnica popular usada em sistemas de recomendação, a Filtragem Colaborativa Baseada em Itens que, ao contrário da filtragem baseada em usuários, foca nas relações entre os itens em vez das relações entre os usuários. Esta abordagem assume que se um usuário gostou de um item no passado, ele pode gostar de itens semelhantes no futuro.

Na Filtragem Colaborativa Baseada em Itens, o sistema analisa também as avaliações feitas por vários usuários e recomenda itens que receberam avaliações semelhantes. A similaridade entre os itens é baseada no comportamento coletivo dos usuários, sem considerar as características específicas dos itens. A principal diferença para a Filtragem

Baseada em Conteúdo é que nessa é comparado diretamente as características dos itens com as preferências anteriores do usuário, recomendando-se itens semelhantes com base em suas descrições, independentemente das avaliações de outros usuários.

Exemplos de Aplicação:

- Amazon: Além da filtragem baseada em usuários, a Amazon também usa filtragem baseada em itens para sugerir produtos relacionados ao produto comprado.
- YouTube: O YouTube recomenda vídeos baseando-se não apenas em quem assistiu o que, mas também em vídeos que são frequentemente assistidos juntos.
- Steam: A plataforma de jogos Steam recomenda jogos analisando quais jogos são frequentemente jogados ou comprados juntos.

Essa técnica ajuda a melhorar a precisão das recomendações e a experiência do usuário ao focar na qualidade das relações entre os itens.

2.3 Recomendação por Regras de Associação

Regras de associação são técnicas utilizadas para identificar padrões e relações entre diferentes itens em grandes volumes de dados. Segundo Kim e Kim (2003), as regras de associação são descritas como padrões que identificam relações frequentes entre itens permitindo prever preferências de usuários com base em itens que costumam ser escolhidos juntos em diferentes níveis de categorização. Essas regras vêm sendo aplicadas em áreas como vendas, sistemas de recomendação de produtos e análise de comportamento de usuários. O principal objetivo das regras de associação é encontrar itens que frequentemente aparecem juntos, o que permite prever comportamentos e preferências futuras dos usuários com base nesses padrões. Por exemplo, se o sistema registra que os usuários sempre que compram pão também compram manteiga, cria-se uma regra que associa esses dois itens e recomenda o produto manteiga caso o usuário compre pão.

No artigo de Cakir e Aras (2012) o autor explica que o principal mecanismo para que essa estratégia de associar elementos funcione são as técnicas de mineração de dados que buscam essas relações entre os diferentes itens comprados por milhares de usuários. O autor explica que os algoritmos buscam identificar relações interessantes entre itens em um conjunto de dados. Essas relações são expressas como regras do tipo "se A, então B", onde, se um determinado item (A) está presente em uma transação, há uma probabilidade de que outro item (B) também esteja presente. O artigo resume os dois principais critérios para gerar essas regras que são chamadas de Suporte e Confiança. Suporte é a proporção de transações que contêm ambos os itens (A e B) e confiança é a proporção de transações que contêm o item A e, dessas, quantas também contêm o item B.

No caso do nosso projeto, de sistema voltado para jogos, poderia-se pensar em recomendar jogos para um usuário com base em suas preferências mas também nas tendências de compra de outros usuários com perfis semelhantes. A ideia seria identificar padrões de comportamento entre os jogadores e criar associações entre os jogos que eles preferem ou compram.

Para se pensar nessa estratégia para nosso projeto precisamos primeiro organizar e representar os dados, por exemplo, como está agora o arquivo CSV, com uma tabela de jogos onde cada linha da planilha tem informações sobre um jogo, incluindo atributos como gênero, console, empresa criadora, e número de compras. Teríamos também uma planilha de Usuários onde cada usuário tem atributos como o console que possuem, gêneros de jogos preferidos, e a lista de desejos (wishList) com jogos que gostariam de comprar.

Exemplo de dados:

Jogo	Gênero	Console	Empresa	Compras
StreetFighter	Ação	PS5	Capcom	500
Elden Ring	RPG	Xbox	FromSoft	300
TombRaider	Aventura	PS5	Microsoft	200

Tabela 1 – Exemplo de dados de jogos

Usuário	Console Preferido	Gênero Preferido	Wishlist
Usuário 1	PS5	Ação	StreetFighter
Usuário 2	Xbox	RPG	Elden Ring
Usuário 3	Switch	Aventura	Zelda

Tabela 2 – Exemplo de dados de usuários

Construção de regras de associação: Com os dados organizados, podemos aplicar a mineração de regras de associação para encontrar padrões entre as preferências dos usuários e os jogos comprados. Por exemplo, usuários que compram jogos de ação para PS5 geralmente também compram jogos de Aventura. Usuários que têm PS5 e gostam de jogos de Ação tendem a adicionar "StreetFighter"à wishList. Essas regras de associação podem ser construídas da seguinte maneira: Se um usuário tem PS5 e gosta de Ação → Recomende também jogos de Aventura do PS5.

2.3.1 Algoritmos usados

- **Apriori**: Um dos algoritmos mais populares para a geração de regras de associação. Ele reduz o número de candidatos usando o princípio de que todos os subconjuntos de um conjunto frequente também devem ser frequentes, filtrando assim grandes quantidades de dados.

- **FP-Growth**: Uma abordagem alternativa ao Apriori, onde uma estrutura compacta chamada de árvore de padrões frequentes (FP-tree) é construída para evitar a geração de um grande número de conjuntos candidatos, tornando o processo de mineração mais eficiente.

2.3.2 Vantagens

- **Personalização**: Os sistemas ajustam as recomendações de acordo com as preferências e o comportamento passado dos usuários, sugerindo itens que são mais relevantes para cada um.
- **Eficiência em Grandes Bases de Dados**: Ao identificar padrões frequentes e gerar regras de forma eficiente, esses sistemas podem processar grandes volumes de dados sem comprometer o desempenho.

2.3.3 Limitações

- **Esparsidade de Dados**: Quando há pouca informação sobre as preferências de um usuário ou quando o número de transações é pequeno, o desempenho das recomendações pode ser prejudicado.
- **Escalabilidade**: Sistemas que utilizam algoritmos como o Apriori podem enfrentar desafios de escalabilidade à medida que a quantidade de itens ou usuários aumenta, o que pode ser mitigado por técnicas como o FP-tree.

2.4 Técnicas Matemáticas Avançadas: Fatoração de Matrizes SVD

A Fatoração de Matrizes Singular Value Decomposition (SVD) (Filho, 2010) é uma técnica matemática usada em muitos campos, incluindo sistemas de recomendação. Ela é considerada uma técnica avançada pois depende de algoritmos matemáticos complexos e de muitas dimensões. Este método é usado para lidar com dados esparsos e que não foram classificados de maneira sistemática ou com erros de classificação. Neste projeto descreveremos brevemente seu uso.

2.4.1 O que é SVD?

A técnica de Decomposição por Valor Singular (SVD) é amplamente usada em sistemas de filtragem colaborativa para identificar características latentes de usuários e itens, ajudando assim a prever preferências individuais e fazer recomendações. Embora o SVD seja popular e disponível em várias plataformas, como o Matlab, sua aplicação direta

em filtragem colaborativa muitas vezes requer ajustes específicos para melhorar a eficácia dos algoritmos de recomendação.

2.4.2 SVD em Sistemas de Recomendação

Em sistemas de recomendação, a matriz A geralmente representa os dados de interação entre usuários e itens, como um usuário que classifica um filme. As linhas representam usuários, e as colunas representam itens. Muitos desses valores são desconhecidos, pois nem todos os usuários classificam todos os itens.

Aqui está como o SVD pode ser usado para preencher essas lacunas (FILHO, 2010):

- **Redução de Dimensionalidade**: O SVD ajuda a reduzir o número de variáveis, mantendo apenas as características mais significativas. Isso simplifica os dados, mantendo ao mesmo tempo a estrutura essencial que captura as relações usuário-item.
- **Previsão de Classificações**: Uma vez que a matriz original A é decomposta, podemos aproximá-la multiplicando as matrizes-filhas. Isso permite prever classificações desconhecidas ao recriar a matriz de usuário-item com as lacunas preenchidas. Essencialmente, isso nos diz como um usuário pode classificar um item que ele nunca avaliou antes.
- **Personalização**: As matrizes resultantes representam, respectivamente, as preferências latentes dos usuários e as características latentes dos itens. Isso pode ser usado para entender melhor as preferências dos usuários e para recomendar itens que se alinham bem com essas preferências.

2.4.3 Desafios e Limitações

- **Esparsidade**: Se a matriz original for muito esparsa (muitos dados faltantes), a eficácia da SVD pode ser limitada.
- **Custo Computacional**: A computação do SVD pode ser intensiva, especialmente para matrizes muito grandes.
- **Dinamismo**: Em um ambiente dinâmico onde novos usuários ou itens estão constantemente sendo adicionados, manter o SVD atualizado pode ser desafiador.

Apesar de sua complexidade, e no contexto de sistemas de recomendação, técnicas como SVD ajudam a criar experiências personalizadas, pois lidam com muitos dados e em muitas dimensões, podendo assim misturar as abordagens de lidar com conteúdo e com as generalidades das recomendações colaborativas.

2.5 Sistemas Híbridos

Para melhorar o desempenho e resolver problemas das técnicas tradicionais de recomendação foram criados métodos híbridos. Esta estratégia envolve a utilização de múltiplos modelos de recomendação, cada um dos quais pode se basear em diferentes conjuntos de dados ou abordagens algorítmicas (como filtragem colaborativa, filtragem baseada em conteúdo, ou métodos baseados em modelo). Esses métodos combinam as melhores partes de duas ou mais técnicas e conseguem fazer recomendações mais precisas, personalizadas e eficientes, superando as limitações de cada técnica usada separadamente. (ADOMAVICIUS; TUZHILIN, 2005; LU et al., 2015)

2.5.1 Abordagem de Mistura (Weighted)

A abordagem ponderada, ou de mistura, envolve a utilização de múltiplos modelos de recomendação, cada um baseado em diferentes conjuntos de dados ou abordagens algorítmicas, como filtragem colaborativa, filtragem baseada em conteúdo ou métodos baseados em modelo (BURKE, 2002). As previsões de cada modelo são combinadas utilizando técnicas de ponderação, que podem incluir uma variedade de métodos e pontuações.

1. Média Simples: O resultado é a média das previsões de todos os modelos. Essa é a forma mais simples de mistura e pode ser eficaz quando todos os modelos são igualmente confiáveis.

2. Média Ponderada: Os resultados de cada modelo são ponderados com base em sua confiabilidade ou desempenho histórico antes de calcular a média. Modelos que historicamente performaram melhor receberão mais peso.

3. Modelos Meta: Um modelo secundário (chamado de meta-modelo) é treinado para otimizar a combinação das previsões dos modelos base. Esse meta-modelo pode ser algo simples como uma regressão linear ou algo mais complexo como uma rede neural.

4. Regras de Decisão: Regras específicas podem ser aplicadas para escolher entre as recomendações dos modelos base em diferentes circunstâncias, como preferir um modelo em particular quando certos critérios ou condições são atendidos.

2.5.2 Abordagem de Sequência (Cascading)

A Abordagem de Sequência, também conhecida como abordagem em cascata, é uma técnica utilizada em sistemas de recomendação híbridos onde múltiplos modelos ou técnicas de recomendação são aplicados sequencialmente. Cada etapa do processo em

cascata refina as recomendações geradas pela etapa anterior, permitindo uma filtragem mais precisa e personalizada (BURKE, 2002).

2.5.3 Métodos de Características Combinadas (Feature Combination)

Os Métodos de Características Combinadas (Feature Combination) são uma série de abordagens que buscam integrar diversas características dos algoritmos em um único modelo. Essa técnica se destaca por combinar diferentes tipos de dados em uma única representação, o que enriquece a análise das preferências dos usuários e aumenta a relevância das recomendações (BURKE, 2002).

Nos métodos de características combinadas, informações de diferentes fontes — como atributos dos itens (gênero, empresa, data de lançamento, desenvolvedor, etc.), dados demográficos dos usuários (idade, gênero, localização), e comportamento de interação (histórico de compras ou avaliações) — são integradas em um vetor único. Esse vetor de características é, então, utilizado por um modelo de aprendizado de máquina ou uma técnica de fatoração de matrizes para encontrar padrões complexos de preferências.

Por exemplo, em um sistema de recomendação de jogos, além de considerar o gênero dos jogos (ação, aventura, RPG), a técnica de características combinadas pode incorporar também a idade do jogador, os jogos que ele possui, a frequência com que joga certos gêneros e até mesmo avaliações que ele deixou em outros jogos. Ao integrar todas essas informações, o modelo se torna capaz de entender melhor as nuances do perfil do usuário e como ele pode reagir a determinados itens.

2.6 Conclusão do capítulo

Neste capítulo, explorou-se as diversas estratégias e algoritmos aplicados a sistemas de recomendação, e deu-se uma visão geral sobre algumas das metodologias de recomendação amplamente utilizadas, como filtragem colaborativa, filtragem baseada em conteúdo, regras de associação e abordagens híbridas. Esses métodos fornecem a base teórica e prática necessária para entender como as recomendações podem ser personalizadas para atender às necessidades dos usuários.

Com base nesse conhecimento, desenvolveu-se um protótipo que aplica esses princípios, integrando técnicas de filtragem simples e similaridade de cossenos. A filtragem inicial permite uma seleção preliminar de jogos baseada em atributos explícitos, como ano de lançamento e gênero, enquanto a técnica de similaridade de cossenos refina essas recomendações, comparando as preferências dos usuários para encontrar jogos similares aos que eles já apreciam.

Essa abordagem de combinação de características (feature combination) mostra

como diferentes técnicas podem ser utilizadas para melhorar a qualidade das recomendações. A implementação prática de nosso protótipo experimenta e nos dá insights para a criação de um sistema mais elaborado. Assim, nossa pesquisa e desenvolvimento nos dá uma melhor compreensão de como teorias avançadas em sistemas de recomendação podem ser melhor aplicadas para beneficiar os usuários finais.

3 Desenvolvimento do Protótipo

3.1 Visão Geral

Em nosso projeto, utilizamos um pouco do método de Características Combinadas para aprimorar a precisão das recomendações. Implementamos uma técnica de filtragem simples, que inicialmente filtra os jogos com base em características explícitas como os jogos que o usuário possui ou os gêneros preferidos do jogador. Caso o jogador já possua o jogo ou se o usuário só quer jogar jogos vintages (jogos antigos lançados nas primeiras décadas da indústria de videogames, de plataformas que não são mais vendidas), o sistema exclui inicialmente vários jogos. Em seguida, aplicou-se a técnica de similaridade por cosseno para comparar dois jogos representados por vetores. Fez-se um primeiro algoritmo com vetores fictícios e ao implementá-lo com os dados vindos do parser de CSV, teve-se muitos problemas e dados incoerentes descritos a seguir. Com isso, adotamos uma nova estratégia de criar uma pontuação baseada na comparação direta entre os dados recebidos pelo usuário e os dados dos jogos, como exemplo, checar se o nome do jogo equivale ao nome de algum jogo presente na lista de jogos possuídos pelo usuário. Esta estratégia de pontuar cada característica permitiu conferir todos os dados vindos do CSV. Por fim, criou-se um algoritmo de distância euclidiana para fazer um comparativo entre as listas criadas.

A figura 2 abaixo representa as etapas pelas quais o projeto passou até chegar na sua versão final.

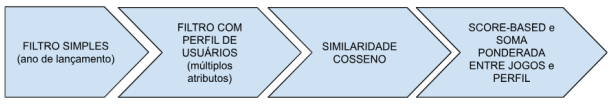


Figura 2 – Fluxograma das etapas de construção do protótipo.

O primeiro passo do protótipo foi um programa que filtrava jogos com base no ano de lançamento. Criamos um programa em C++ com uma estrutura baseada em uma classe denominada Game que contém em suas variáveis as informações de um determinado jogo (nome do jogo, data de publicação, nome da empresa que publicou, plataforma do jogo e palavras chaves sobre detalhes do jogo), conforme ilustrado no diagrama de classes da Figura 3.

Em um primeiro momento estruturou-se o protótipo em três arquivos de programa: o arquivo da classe jogo (Game.cpp), o arquivo cabeçalho da classe (Game.h) e o arquivo principal do programa (main.cpp). Para efeito de teste, se criou uma lista de 5 jogos a partir das classes e criamos uma função que inicialmente listava (print) todos os jogos.

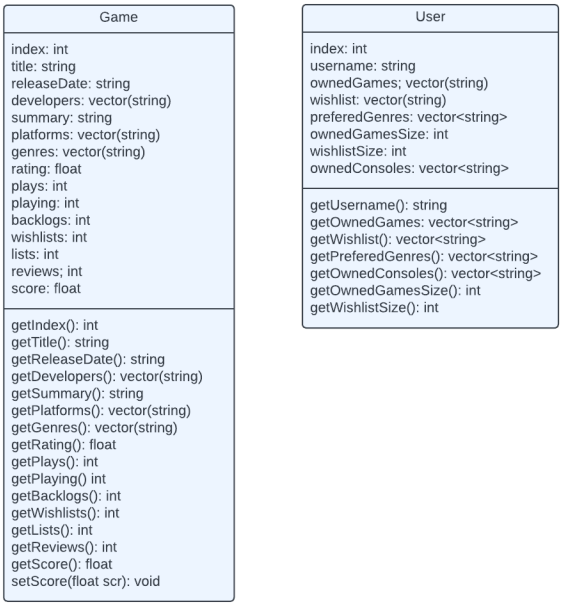


Figura 3 – Diagrama de classes das classes utilizadas

Após isto, criou-se um parser para aumentar a quantidade de jogos que se pode comparar, sem que fosse necessário manualmente inserir os jogos, e pudesse se ler os jogos de uma grande base de dados.

3.2 Base de Dados

3.2.1 Introdução

Em seguida, procurou-se uma base de dados que pudesse fornecer de forma confiável as informações sobre jogos digitais já lançados, e montou-se um programa que pudesse ler e analisar esta base de dados. Foi encontrada uma plataforma on-line chamada Kaggle (kaggle.com), que é uma plataforma que oferece uma gama de ferramentas e recursos para cientistas de dados e de machine learning. Fundada em 2010 e adquirida pelo Google em 2017, ela se tornou um ponto de encontro para profissionais e amadores da área de ciência de dados. A plataforma também é conhecida como um repositório de bases de dados acessíveis de vários temas diferentes. Assim, procurando pelo termo "popular video games", buscou-se uma base de dados generalista que pudesse servir de exemplo. Encontrou-se então uma base criada por Matheus Fonseca Chaves chamada "Popular Video Games" que pode ser encontrada no link a seguir (https://www.kaggle.com/datasets/matheusfonsecachaves/popular-video-games). A base de dados está no formato CSV cujo arquivo se chama "backloggd_games.csv". Criou-se as variáveis internas da classe Jogo com base nos atributos extraídos das colunas desse CSV. Em uma sessão a seguir, descreve-se em detalhes o arquivo e suas colunas e como foi adaptado o arquivo ao programa. Para a etapa seguinte do projeto, foi visto que

seria necessário uma série de funções capazes de ler e guardar as informações dos jogos em variáveis. É uma função tradicionalmente chamada de Parser, que é um software ou uma função que interpreta e processa arquivos de texto, neste caso em formato CSV. Nessa versão do projeto, o grupo decidiu não usar todas as colunas do CSV. Para o programa C++ melhor ler os dados do arquivo, decidiu converter o arquivo CSV original em um arquivo em formato JSON. Para isso, escreveu se um programa em python chamado csvToJson.py que pega, por meio da biblioteca python csv, cada linha do arquivo (em que estava descrito um único jogo) e o transforma em uma lista em que cada item é um dicionário representando um jogo, e dentro destes dicionários, cada informação sobre o jogo é um sub-item, com a sua chave sendo o nome da informação (título, data de lançamento, desenvolvedores, dentre outros).

3.3 Formato da Base de Dados

O arquivo CSV usado para banco de dados, e o arquivo JSON criado a partir dele, consistem nas colunas:

- ID do jogo
- Nome do jogo
- Data de lançamento
- Desenvolvedores
- Sumário do jogo
- Plataformas em que o jogo está disponível
- Gêneros do jogo
- Média de avaliações de 0 a 5
- Número de vezes que foi jogado
- Número de pessoas jogando no momento da retirada dos dados
- Número de pessoas com o jogo em sua lista de desejos
- Número de listas em que o jogo está
- Número de avaliações escritas que o jogo possui

Exemplo de código do CSV:

```
1. ID, Title, Release\_Date, Developers, "Summary", Platforms, Genres, Plays, Playing, Backlogs,
Wishlist, Lists, Reviews}
2. 0,Elden Ring,Feb 25, 2022,[FromSoftware, Bandai Namco Entertainment],"Elden Ring is a
fantasy, action and open world game with RPG elements such as stats, weapons and spells.
Rise, Tarnished, and be guided by grace to brandish the power of the Elden Ring and
become an Elden Lord in the Lands Between.",
[Windows PC, PlayStation 4, Xbox One, PlayStation 5, Xbox Series],
[Adventure, RPG],4.5,21K,4.1K,5.6K,5.5K,4.6K,3K
```

Exemplo de código do JSON:

```
{
    "games": [
        {
            "Index": "0",
            "Title": "Elden Ring",
            "Release_Date": "Feb 25, 2022",
             "Developers": "['FromSoftware', 'Bandai Namco Entertainment']",
             "Summary": "Elden Ring is a fantasy, action and open world game with RPG
             elements such as stats, weapons and spells. Rise, Tarnished, and be
             guided by grace to brandish the power of the Elden Ring and become an
             Elden Lord in the Lands Between.",
            "Platforms": "['Windows PC', 'PlayStation 4', 'Xbox One', 'PlayStation 5',
            'Xbox Series']",
            "Genres": "['Adventure', 'RPG']",
            "Rating": "4.5",
            "Plays": "21K",
            "Playing": "4.1K",
            "Backlogs": "5.6K",
            "Wishlist": "5.5K",
            "Lists": "4.6K",
            "Reviews": "3K"
        },
```

3.4 Implementação

3.4.1 Estrutura do Programa

Começou-se, portanto, o projeto em um versão inicial que contava com a seguinte estrutura:

Um arquivo main, que continha 5 jogos codificados diretamente no programa, cada um um vetor que recebia as informações dos jogos como nome, ano de lançamento, tags e empresa desenvolvedora, além de uma função para imprimir os jogos e suas informações, e um simples filtro com base no ano de lançamento.

Em seguida, junto com o uso dessas bibliotecas C++, criou-se o parser para ler o arquivo CSV, que foi codificado no arquivo CSVReader.cpp e seu header (CSVReader.h). Este arquivo contém as funções readCSV, que lê o CSV propriamente, parseCSVLine, que analisa cada linha do CSV, strToVec, que transforma uma string representando uma lista de itens em um vetor, strToNum, que transforma uma string com a letra K representando um número em milhares em uma int, e strToDate, que transforma uma string contendo uma data em um tipo timepoint.

Montamos os arquivos de classe Game e User para guardar as informações dos jogos retirados do CSV, e dos usuários criados para o programa. Também movemos as funções de impressão do arquivo main para um arquivo próprio chamado printerFunctions.cpp. O codigo do arquivo CSVReader.cpp está apresentado em anexo no final do texto.

3.4.2 Classe Game e suas variáveis

A classe "Game"foi criada para receber os dados extraídos do arquivo CSV via o arquivo CSVReader. Esta classe encapsula uma variedade de atributos essenciais que caracterizam cada jogo. Entre eles estão variáveis como título, data

de lançamento, desenvolvedores, resumo do jogo, plataformas disponíveis, gêneros, e a avaliação (rating). Além dessas informações, a classe também contém métricas de engajamento e popularidade, como o número de avaliações recebidas, a frequência de jogos executados, o número atual de jogadores, e a inclusão do jogo em listas de desejos e backlogs (quando um jogo é comprado mas ainda não foi inicializado).

Adicionalmente, a classe "Game"encapsula cada jogo listado no arquivo CSV, que é utilizado como banco de dados. Este arquivo contém registros de cada jogo, facilitando o gerenciamento e a recuperação de informações.

A classe está codificada em Listagem 3 - Classe Game.

3.4.3 Classe User e suas variáveis

Para a segunda etapa, focamos em criar o algoritmo de comparação entre o jogo e o usuário. Para isso, a classe User foi criada para encapsular os dados de tal usuário. Em contrapartida aos jogos, os usuários não são retirados de nenhum arquivo pré-definido, mas codificados diretamente no programa para fins de teste. Na potencial continuação desse projeto, haveria uma maneira das pessoas criarem novos usuários, e seria possível fazer login no sistema com base em seu usuário.

A classe está codificada em Listagem 3 - Classe User.

3.4.4 Algoritmos de Filtragem e de similaridade

O algoritmo de filtragem foi iniciado no arquivo GameFilter.cpp. A função filterByUser recebe uma instância do User e uma lista de Games. Começamos a desenvolver essa função, mas no final decidimos integrá-la no sistema de comparação, presente no arquivo comparisonFunctions.cpp.

O rascunho da classe GameFilter está localizado no Apêndice I - Listagem 5 - GameFilter.

3.4.4.1 Algoritmos de similaridade

Para o algoritmo de similaridade, inicialmente programamos um algoritmo que utiliza a Similaridade de Cosseno para comparar dois vetores. Ele foi baseado na fórmula apresentada no tópico 2.1.3.1, “Similaridade de Cosseno”, e cujo código se encontra no Apêndice I.

Como o algoritmo de Similaridade de Cosseno recebe dois vetores numéricos, que precisam ser codificados antes de podermos utilizá-los na fórmula, necessitamos traduzir os jogos para valores numéricos e para isso deveríamos comparar os jogos com os parâmetros do usuário. Para isso escolhemos parâmetros coerentes como jogos desejados (wishlist), jogos que já possui (ownedGames) e plataformas que o usuário possui ou mesmo gêneros que prefere. Assim excluímos nessa comparação características como ano de lançamento do jogo, desenvolvedores, entre outros. Esta tradução dos jogos para valores numéricos juntamente aos problemas ligados aos dados incoerentes do parser gerou vários erros, e por isso decidiu-se montar um algoritmo com escores.

O algoritmo de comparação baseado em escores, compara as informações similares entre o jogo e o usuário. O resultado da comparação deveria ser um score final que nos daria o quão similar seria as características do jogo com as desejadas pelo usuário. O resultado das comparações nos daria uma série de valores que, no final, seriam somados na própria classe Jogo como uma variável a mais, chamada score de compatibilidade. Com esse score, pode-se criar uma lista em ordem decrescente de scores de compatibilidade com jogos mais relevantes para o usuário estando no topo da lista. Caso o score final de dois jogos fossem idênticos, também ordenou-se a lista decrescentemente com base na variável “rating” de cada jogo, que é a média das notas dadas por jogadores ao jogo, externamente ao sistema. A informação do rating foi obtida pela base de dados.

Teve-se também problemas ao criar esse comparativo, principalmente com os dados extraídos do CSV, com alguns jogos recebendo resultados muito semelhantes uns aos outros. Estava-se também tendo resultados binários muito frequentemente, pois comparamos informações do tipo se o usuário possui o jogo ou não, ou se ele gosta de um certo gênero ou não. Para isso criou-se formas de somar a pontuação caso a coluna de gênero (aventura, ação, plataforma) coincidisse várias vezes entre o usuário e o jogo. Neste caso, um exemplo disto seria de, se um jogo possuísse os gêneros plataforma e estratégia. Se um usuário disse que gostava somente de jogos de estratégia, então o jogo receberia meio ponto no quesito de gêneros. No caso de outro usuário, que disse gostar tanto de jogos de estratégia quanto de jogos de plataforma, o jogo receberia um ponto inteiro no quesito de gêneros.

Outra estratégia era criar pesos entre os diversos tipos de características. Por exemplo, seria mais importante valorizar a correspondência entre o gênero de um jogo do que a quantidade de plataformas em que o jogo está disponível.

Assim deve-se em uma próxima etapa experimentar as diversas estratégias de codificação dos dados extraídos e comparados com o usuário.

3.4.4.2 Algoritmos de comparação (comparisonFunctions.cpp)

Esse arquivo é onde está o algoritmo de similaridade final, e que integra também a estratégia inicial de filtragem. Ele reconhece se o jogo dado a ele está na lista de jogos possuídos ou na lista de desejos do usuário. Ele também reconhece a quantidade de gêneros que o jogo compartilha com os gêneros preferidos do usuário. Por último, ele vê se o jogo está disponível em algum dos consoles de que o usuário é dono.

O resultado deveria ser um score final que soma todas essas características comparadas e seria utilizado para ordenar os jogos da lista de jogos retirados do JSON para recomendação.

O arquivo comparisonFunctions.cpp possui a função compareUserGames(), que compara os atributos similares do usuário inserido e os jogos dados, para então calcular um escore final para uso na filtragem de jogos. A função compara o jogo atual com os jogos da lista de posse do jogador, da lista de desejos do jogador, as plataformas disponíveis do jogo, e os gêneros do jogo.

O pseudocódigo da função compareUserGames() pode ser visto abaixo:

```
compareUserGame(user, game)
finalScore = 0
// Descobrir se usuario possui jogo
vector ownedGames = user.getOwnedGames()
found = false
for (gameTitle in ownedGames)
if gameTitle == jogo.getTitle()
                found = true
                break
if (found = true)
        return -1
else
        finalScore++
// Descobrir se jogo esta em alguma plataforma possuida pelo usuario
gameOnOwnedPlatform = false
for (userPlatforms in user.getOwnedConsoles())
        for (gamePlatforms in jogo.getPlatforms())
                if (userPlatform == gamePlatform)
                        gameOnOwnedPlatform = true
                        break
        if (gameOnOwnedPlatform) break
if (not gameOnOwnedPlatform)
        return -1
else
        finalScore++
// Descobrir se jogo esta na wishlist de usuario
for (gameWL in user.getWishlist())
        if (gameWL == jogo.getTitle())
                finalScore++
// Comparar generos preferidos
numGenresFound = 0
for (userGenre in user.getPreferedGenres())
        for (gameGenre in jogo.getGenres())
                if (userGenre == gameGenre)
                        numGenresFound++
gameGenreSize = jogo.getGenres().size()
genreScore = 0
if (gameGenreSize > 0)
        genreScore = numGenresFound / gameGenreSize
finalScore = genreScore + finalScore
return finalScore
```

3.4.4.3 Algoritmo de Filtragem (listFunctions.cpp)

O algoritmo de filtragem está presente no arquivo listFunctions.cpp. Após o cálculo do escore de compatibilidade, a função addGameToList() pega o score de cada jogo recebido e adiciona ele ou não a uma lista de 100 jogos. O jogo é adicionado na posição de ordem decrescente do escore de compatibilidade calculado. Se o escore de compatibilidade é igual, também se é comparado o escore de avaliação geral do jogo (rating). Caso um jogo esteja na posição 101 da lista, ele é retirado.

O pseudocódigo da função addGameToList() está disponível abaixo:

```
addGameToList(novoJogo, vector gameList)
    if (novoJogo.getScore > 0)
        // Verificar se o jogo ja esta na lista
        it = find_if(gameList.begin(), gameList.end(), (jogo) {
            return jogo.getTitle() == novoJogo.getTitle(); });
        if (it == gameList.end())
                pos = find_if(gameList.begin(), gameList.end(), (jogo) {
                    return jogo.getScore() < novoJogo.getScore() ||
                    (jogo.getScore() == novoJogo.getScore() && jogo.getRating()
                    < novoJogo.getRating());
                });
            // Inserir o novo jogo na posicao encontrada
            gameList.insert(pos, novoJogo)
        if (gameList.size() > 100)
            gameList.pop_back()
```

3.4.5 Estrutura Final

Ao final do projeto, houve um grande aumento no número de arquivos de código. As diversas funções do programa foram divididas em grupos similares, cada um no seu arquivo separado. Assim, ficou-se com arquivos de funções referentes ao CGI, funções de impressão para o terminal ou para a página HTML, funções que mexem em listas, funções de leitura de JSON, dentre outras. No HTML criou-se um formulário que alimenta o programa com dados preenchidos pelo usuário.

A figura 4, abaixo, mostra um diagrama de sequência simplificado do projeto final, e como os diferentes arquivos C++ interagem uns com os outros por meio das chamadas de funções.

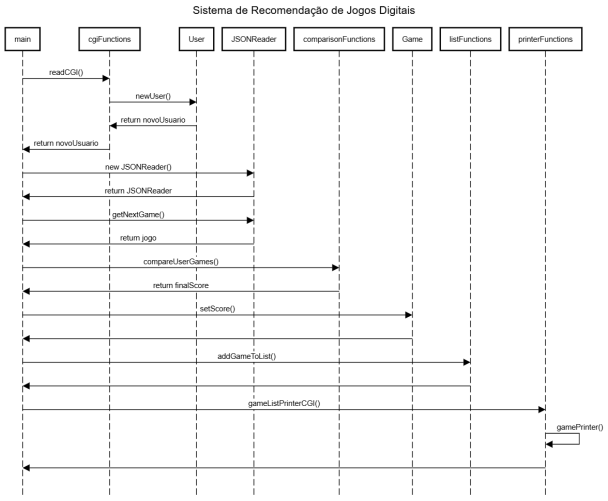


Figura 4 – Diagrama de sequência do projeto

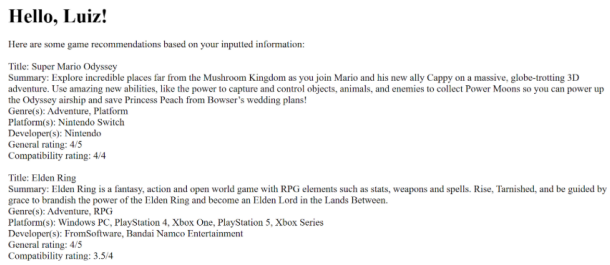
4 Validação de Resultados

Neste capítulo, será descrita a validação dos resultados obtidos por meio do programa, tanto internamente, comparando elementos da base de dados entre si para garantir a coerência de resultados, quanto externa, utilizando usuários reais para garantir que os resultados condizem com o que seria esperado deles.

4.1 Validação Interna

Em relação à validação interna (validação feita com usuários pré-programados), percebeu-se que o algoritmo estava sendo coerente. Ele retorna uma lista com que obedece às regras programadas.

Na figura 5, está demonstrado uma saída tradicional do programa de recomendação.


Hello, Luiz!

Here are some game recommendations based on your inputted information:

Title: Super Mario Odyssey
Summary: Explore incredible places far from the Mushroom Kingdom as you join Mario and his new ally Cappy on a massive, globe-trotting 3D adventure. Use amazing new abilities, like the power to capture and control objects, animals, and enemies to collect Power Moons so you can power up the Odyssey airship and save Princess Peach from Bowser's wedding plans!
Genre(s): Adventure, Platform
Platform(s): Nintendo Switch
Developer(s): Nintendo
General rating: 4/5
Compatibility rating: 4/4

Title: Elden Ring
Summary: Elden Ring is a fantasy, action and open world game with RPG elements such as stats, weapons and spells. Rise, Tarnished, and be guided by grace to brandish the power of the Elden Ring and become an Elden Lord in the Lands Between.
Genre(s): Adventure, RPG
Platform(s): Windows PC, PlayStation 4, Xbox One, PlayStation 5, Xbox Series
Developer(s): FromSoftware, Bandai Namco Entertainment
General rating: 4/5
Compatibility rating: 3.5/4

Figura 5 – Exemplo de saída do programa

4.2 Validação Externa

4.2.1 Testes

Para a validação externa (validação feita com usuários reais), foi tentado disponibilizar o sistema de recomendação em formato de aplicativo web em um servidor cloud para ser acessado remotamente pelos usuários. No entanto, devido a dificuldades na habilidade de rodar um programa com uso de CGI em servidores externos, optou-se por iniciar os testes com usuários de maneira remota.

Para isto, criou-se um formulário de avaliação na plataforma Google Forms com perguntas referentes às entradas do programa (Nome de usuário, Consoles Possuídos, Jogos Possuídos, Lista de Desejos de Jogos e Gêneros Preferidos de Jogos), e com as respostas deste formulário (figura 6), o programa foi rodado para cada usuário. O formulário 1 pode ser encontrado por completo no Apêndice II.

A saída do programa (Figura 7) referente a cada usuário foi mandada por email para seus respectivos donos, e por último, pediu-se para preencherem um segundo formulário de feedback sobre as respostas. Este segundo formulário pede aos usuários para dar uma nota em uma escala de 1 a 5 para se eles gostaram da ideia de um recomendador de jogos, e para o quão úteis eles acharam as recomendações de jogos dadas pelo programa. Por último, perguntou se, por escrito, quais as melhorias que eles achavam que o recomendado poderia ter. O formulário 2 pode ser encontrado por completo no Apêndice II.

Até o momento de escrita, foram recebidos 39 usuários para os testes.

4.2.2 Resultados

Dos 39 testes feitos, 8 usuários preencheram o segundo formulário de feedback, obtendo-se uma nota de 4 de 5 em relação à pergunta se os usuários gostaram da idéia de um recomendador de jogos, como pode se ver na figura 8.

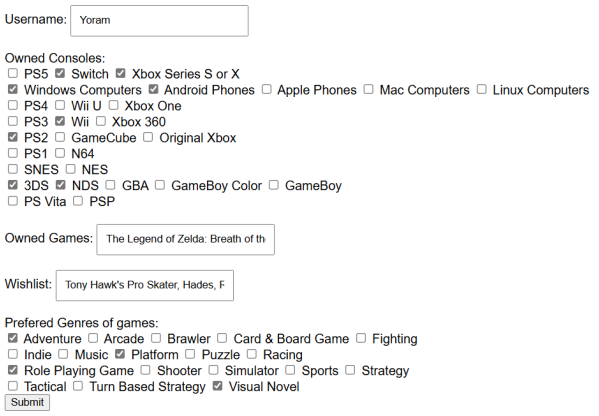


Figura 6 – Exemplo de entrada do programa

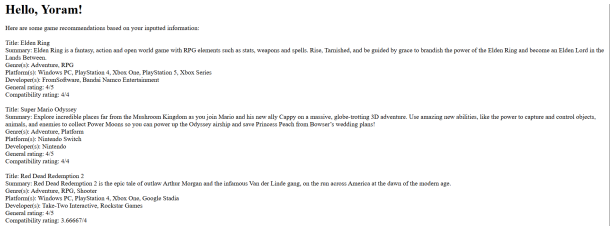


Figura 7 – Outro exemplo de saída do programa

Figura 8 – Gráfico de utilidade de recomendador

Na segunda pergunta feita no formulário de feedback, se os usuários acharam as recomendações do recomendador úteis, obteve-se uma nota de 3.625 de 5. Isso pode ser visto na figura 9.

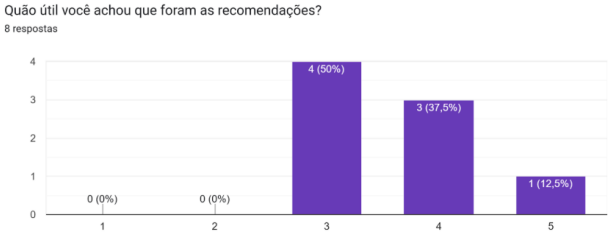


Figura 9 – Gráfico de satisfação

Algumas ideias de melhorias recebidas dos testadores sobre o programa foram:

- A possibilidade de mostrar mais jogos por rodadas de recomendação;
- Mostrar juntamente com as informações os preços atuais dos jogos;

- Colocar um link para a página de compra de uma loja digital do jogo;
- Não mostrar tanto os jogos que já foram colocados como jogos da lista de desejos.

5 Considerações Finais

No começo, implementamos um filtro simples para testar a estrutura básica do sistema. Esse filtro era baseado no ano de lançamento dos jogos, o que nos permitiu depois continuar a construir o algoritmo. Apesar de ser um filtro simples, foi importante para montar a arquitetura do protótipo e entender melhor como organizar os dados e implementar funcionalidades mais complexas.

Durante o desenvolvimento do nosso protótipo de sistema de recomendação de jogos digitais, enfrentamos uma série de desafios e identificamos várias oportunidades de melhoria. Aqui, discutimos as principais dificuldades, os aprendizados e as perspectivas para futuras iterações.

5.1 Dificuldades

5.1.1 Algoritmo de Similaridade

Para começar, desenvolveu-se um algoritmo de similaridade para melhorar as recomendações. O grupo utilizou a Similaridade de Cosseno, que é uma métrica para comparar vetores de características dos jogos. A implementação deste algoritmo revelou alguns problemas com as saídas, que deveriam ser vetores extraídos das comparações entre o perfil do usuário e as características dos jogos com uma variedade de valores, mas que acabaram todos sendo de valores 0 ou 1.

Tivemos problemas para identificar o problema relativos aos valores incoerentes das saídas. A primeira hipótese é que os valores lidos pelo parser poderiam estar incorretos. Então decidiu-se, por um lado, trabalhar com um arquivo JSON ao invés de CVS para a base de dados, além de reverter algumas partes do código para versões anteriores, e redesenhar o algoritmo de similaridade. Ao final, utilizou-se um algoritmo de comparação de características e soma de pontuação acumulada de critérios múltiplos, em vez de uma métrica geométrica clássica como distância euclidiana, cosseno ou Jaccard.

5.1.2 Desafios com Inputs e Ajustes Necessários

Um dos principais desafios foi garantir que os vetores de características fossem precisos e refletissem fielmente tanto os dados objetivos (como gênero, desenvolvedor, data de lançamento) quanto às preferências subjetivas do usuário. Percebemos que o algoritmo precisa ser ajustado (ou melhor, refinado) para equilibrar essas duas dimensões e proporcionar recomendações mais relevantes.

5.2 Aprendizados

Durante o projeto, aprendeu-se a melhor programar em C++ e melhor depurar a linguagem de programação.

Além do mais, foi adquirido um maior conhecimento em uso da biblioteca cgicc, que facilita a impressão de HTML por meio de técnicas de Common Gateway Interface para a linguagem de C++.

Ademais, foi possível observar os impactos das escolhas tecnológicas sobre o resultado do projeto. Por exemplo, foi por causa da linguagem de programação C++ que usou-se no projeto, houveram dificuldades em colocar o projeto para rodar em um servidor externo.

Por último, o grupo também ganhou conhecimento dos diversos modelos de sistema de recomendação e de diversos algoritmos de filtragem utilizados nele, vantagens e limitações de cada categoria, e os múltiplos tipos de uso de cada tipo de algoritmo.

5.3 Melhorias Futuras

Para os próximos passos desse projeto, seria interessante continuar o desenvolvimento do algoritmo de similaridade para fazer a comparação de um maior número de características, como por exemplo os desenvolvedores de jogos favoritos do

usuário, ou focar em jogos mais recentes ou mais antigos.

Deve-se também alterar a estrutura do programa para diminuir a quantidade de jogos recomendados que já estão presentes na wishlist recebida do usuário, já que o usuário já tem conhecimento e interesse nestes jogos e não precisa deles como recomendação.

Ademais, seria necessário escolher uma nova base de dados para o programa, já que a base de dados utilizada no projeto inclui também bastante jogos duplicados, ou jogos que não são comercialmente disponíveis (por exemplo, modificações feitas por fãs) e não inclui todas as informações que seriam ideais para o display do jogo para os usuários (por exemplo, não há uma imagem do jogo, nem seu preço atual). Idealmente, se implementaria uma conexão entre o programa e uma API que receba informações de jogos direto da loja virtual em que ela pode ser comprada (Steam, Nintendo eShop, PlayStation store ou Microsoft Store).

Além disto seria possível aumentar a complexidade do algoritmo para refinar ainda mais a nota final de recomendação do jogo, como exemplo adicionando características das filtragens colaborativas para melhor recomendar jogos. Uma outra ideia interessante seria a implementação de técnicas de aprendizado de máquina e inteligência artificial para identificar padrões mais complexos nos dados e melhorar as previsões de preferências dos usuários.

Mais uma mudança possível seria a criação de contas de usuário com login e senha, para que o usuário possa salvar sua coleção de jogos e wishlist em caso de múltiplos usos do programa.

Por último, o programa também necessitaria da criação de um User Experience mais condizente com um aplicativo moderno: o uso de um servidor para o usuário final poder usá-lo direto remotamente, o uso de cores e técnicas de design para formatar tanto a página de entrada quanto a página de respostas de maneira mais agradável para os usuários.

5.4 Conclusão

Apesar das dificuldades encontradas, o desenvolvimento do protótipo foi uma experiência rica em aprendizado. Conseguimos criar uma base sólida para um sistema de recomendação que, com ajustes e refinamentos, tem potencial para ser ainda mais eficaz. Nosso próximo passo seria a integração de métodos mais avançados, como os detalhados acima, explorar novas fontes de dados para continuar aprimorando o sistema e experimentar e criar diferentes interfaces de acesso ao sistemas pelo público.

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*, v. 17, n. 6, p. 734–749, 2005. Citado na página 31.

AGGARWAL, C. *Recommender Systems: The Textbook*. [S.l.: s.n.], 2016. Citado na página 22.

ANWAR, S. M. et al. A game recommender system using collaborative filtering (GAMBIT). In: *2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. [S.l.: s.n.], 2017. p. 328–332. Citado 2 vezes nas páginas 15 e 17.

BARTH, F. Modelando o perfil do usuário para a construção de sistemas de recomendação: um estudo teórico e estado da arte. *Revista de Sistemas de Informação da FSMA*, v. 6, p. 59–71, 2010. Citado na página 19.

BENEDITO, N. *Economia de atenção e a manutenção do interesse no mundo 6.0*. Revista Valor, 2024. Disponível em: <https://valor.globo.com/patrocinado/pressworks/noticia/2024/06/03/economia-de-atencao-e-a-manutencao-do-interesse-no-mundo-60.ghtml>. Citado na página 16.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006. Citado na página 23.

BURKE, R. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, v. 12, p. 331–370, 2002. Citado 2 vezes nas páginas 31 e 32.

CAKIR, O.; ARAS, M. E. A Recommendation Engine by Using Association Rules. In: *Procedia - Social and Behavioral Sciences*. [S.l.]: Elsevier Ltd., 2012. v. 62, p. 452–456. Citado 2 vezes nas páginas 17 e 27.

CHAVES, M. *Popular Video Games*. Kaggle, 2023. Disponível em: <https://www.kaggle.com/datasets/matheusfonsecachaves/popular-video-games>. Citado na página 18.

CHEUQUE, G.; GUZMáN, J.; PARRA, D. Recommender Systems for Online Video Game Platforms: The Case of STEAM. In: *Companion Proceedings of The 2019 World Wide Web Conference*. New York, NY, USA: Association for Computing Machinery, 2019. (WWW '19), p. 763–771. ISBN 978-1-4503-6675-5. Event-place: San Francisco, USA. Disponível em: <https://doi.org/10.1145/3308560.3316457>. Citado na página 15.

FILHO, J. B. A. P. *Um algoritmo de filtragem colaborativa baseado em SVD*. Tese (Dissertação de Mestrado) — Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-graduação em Ciência da Computação, Florianópolis, 2010. Citado 4 vezes nas páginas 11, 24, 25 e 30.

HEFEZ, A.; VILLELA, M. L. T. *Códigos corretores de erros*. [S.l.]: Instituto de Matematica Pura e Aplicada, 2008. Citado na página 23.

KIM, C.; KIM, J. A Recommendation Algorithm Using Multi-Level Association Rules. In: *Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI'03)*. IEEE, 2003. p. 524–527. Backup Publisher: IEEE. Disponível em: <https://doi.org/10.1109/WI.2003.1241328>. Citado 2 vezes nas páginas 17 e 27.

LIMA, D. *Mais de 14 mil jogos foram lançados no Steam em 2023*. 2024. Disponível em: <https://www.theenemy.com.br/pc/steam-jogos-2023-milhares>. Citado na página 16.

LOPS, P.; GEMMIS, M. D.; SEMERARO, G. Content-based Recommender Systems: State of the Art and Trends. In: RICCI, F. et al. (Ed.). *Recommender Systems Handbook*. [S.l.]: Springer US, 2011. p. 73–105. Citado 2 vezes nas páginas 17 e 18.

LU, J. et al. Recommender System Application Developments: A Survey. *Decision Support Systems*, v. 74, p. 12–32, 2015. Citado na página 31.

Lü, L. et al. Recommender systems. *Physics Reports*, v. 519, n. 1, p. 1–49, out. 2012. ISSN 0370-1573. Number: 1 Publisher: Elsevier BV. Disponível em: <http://dx.doi.org/10.1016/j.physrep.2012.02.006>. Citado na página 17.

MATOS, D. *Data Science na Indústria de Videogames*. 2022. Publication Title: Ciência de Dados. Disponível em: <https://www.cienciaedados.com/data-science-na-industria-de-videogames/>. Citado na página 19.

MENEGHESSO, C. *Códigos Corretores de Erros*. Tese (PhD Thesis) — Universidade Federal de Sao Carlos, 2012. Citado na página 23.

MUREL, J.; KAVLAKOGLU, E. *O que é filtragem colaborativa? | IBM*. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/collaborative-filtering>. Citado 3 vezes nas páginas 24, 25 e 26.

Spotify AB. *Spotify*. Disponível em: <https://www.spotify.com>. Citado na página 15.

SteamDB. *Steam Game Release Summary by Year*. 2023. Disponível em: <https://steamdb.info/stats/releases/>. Citado na página 16.

THORAT, P. B.; GOUDAR, R. M.; BARVE, S. Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System. *International Journal of Computer Applications*, v. 110, n. 4, 2015. ISSN 0975-8887. Number: 4 Publisher: MIT Academy of Engineering, Pune India. Citado 2 vezes nas páginas 17 e 24.

Valve Corporation. *Steam*. Disponível em: <https://store.steampowered.com>. Citado na página 15.

Anexos

ANEXO A – Código-fonte do protótipo

A.1 SimilaridadeCosseno.cpp

```
#include "SimilaridadeCosseno.h"

double calcularSimilaridadeCosseno(const std::vector<double>& A,
const std::vector<double>& B) {
    double produtoEscalar = 0.0;
    double normaA = 0.0;
    double normaB = 0.0;


    if (A.size() != B.size()){
        return -1;
    }


    for (size_t i = 0; i < A.size(); ++i) {
        produtoEscalar += A[i] * B[i];
        normaA += A[i] * A[i];
        normaB += B[i] * B[i];
    }


    normaA = std::sqrt(normaA);
    normaB = std::sqrt(normaB);


return produtoEscalar / (normaA * normaB);
}
```

A.2 comparisonFunctions.cpp

```
#include "comparisonFunctions.h"
#include "generalFunctions.h"


extern bool isTest;


float compareUserGames(User user, Game jogo) {
    vector<float> scores;
    float finalScore = 0;


    // Comparar jogos possuidos
    bool found = false;
    vector<string> ownGames = user.getOwnedGames();


    for (const string& gameTitle : ownGames){
        if (convertToLower(gameTitle) == convertToLower(jogo.getTitle())){
```

```
            found = true;
        }
    }


    if (found == true){
        debug(user.getUsername() + " ja possui o jogo " + jogo.getTitle() + "\n",
        isTest);
        scores.push_back(-1);
        return -1;
    }
    else{
        scores.push_back(1);
        finalScore++;
    }


    // Comparar plataformas possuidas
    bool gameOnOwnedPlatform = false;


    for (const string& userPlat : user.getOwnedConsoles()) {
        for (const string& gamePlat : jogo.getPlatforms()){
            if (convertToLower(userPlat) == convertToLower(gamePlat)){
                gameOnOwnedPlatform = true;
                break;
            }
        }
        if (gameOnOwnedPlatform){break;}
    }
    if (!gameOnOwnedPlatform){
        debug("Jogo nao disponivel nas plataformas de " + user.getUsername(), isTest);
        debug("Score total de " + jogo.getTitle() + ": " + to_string(finalScore) + "\n",
        isTest);
        return -1;
    }
    else{
        scores.push_back(1.0);
        finalScore++;
    }


    // Comparar wishlist
    for (const string& gameWL : user.getWishlist()){
        if (convertToLower(gameWL) == convertToLower(jogo.getTitle())){
            finalScore++;
            scores.push_back(1.0);
        }
        else{
            scores.push_back(0.0);
        }
    }


    // Comparar generos preferidos
    float numGenresFound = 0;
    for (const string& userGenre : user.getPreferedGenres()) {
        for (const string& gameGenre : jogo.getGenres()){
            if (convertToLower(userGenre) == convertToLower(gameGenre)){
                numGenresFound++;
```

```
            }
        }
    }
    int gameGenreSize = jogo.getGenres().size();
    float genreScore = 0;
    if (gameGenreSize > 0) {
        genreScore = numGenresFound / gameGenreSize;
        debug("Numero de generos iguais: " + to_string(numGenresFound), isTest);
        debug("Numero de generos do jogo: " + to_string(gameGenreSize), isTest);
        debug("Score genero: " + to_string(genreScore) + "\n", isTest);
    }
    finalScore = genreScore + finalScore;
    scores.push_back(genreScore);
    debug("Score jogo possuido: " + to_string(scores[0]) + "; Score console: " +
    to_string(scores[1]) + ";", isTest);
    debug("Score lista de desejos: " + to_string(scores[2]) + "; Score Genero: " +
    to_string(scores[3]) + ";", isTest);
    debug("Score total de " + jogo.getTitle() + ": " + to_string(finalScore) + "\n",
    isTest);
    return finalScore;
}
```

A.3 Classe Game (Game.cpp)

```
#include "Game.h"

Game::Game(int i, string ti, string rd, vector<string> dev, string sum,
vector<string> plat, vector<string> gen, float rat, int pl, int pl2, int bkg,
int wsh, int lst, int rv)
{
    index = i;
    title = ti;
    releaseDate = rd;
    developers = dev;
    summary = sum;
    platforms = plat;
    genres = gen;
    rating = rat;
    plays = pl;
    playing = pl2;
    backlogs = bkg;
    wishlists = wsh;
    lists = lst;
    reviews = rv;
    score = 0;
}


int  Game::getIndex(){ return index; };
string Game::getTitle() const { return title; };
string Game::getReleaseDate(){ return releaseDate; };
vector<string> Game::getDevelopers(){ return developers; };
string Game::getSummary(){ return summary; };
vector<string> Game::getPlatforms(){ return platforms; };
vector<string> Game::getGenres(){ return genres; };
float Game::getRating() const { return rating; };
int Game::getPlays(){ return plays; };
```

```
int Game::getPlaying(){ return playing; };
int Game::getBacklogs(){ return backlogs; };
int Game::getWishlists(){ return wishlists; };
int Game::getLists(){ return lists; };
int Game::getReviews(){ return reviews; };
float Game::getScore() const { return score; };
void Game::setScore(float scr) { score = scr; };
```

A.4 Classe User (User.cpp)

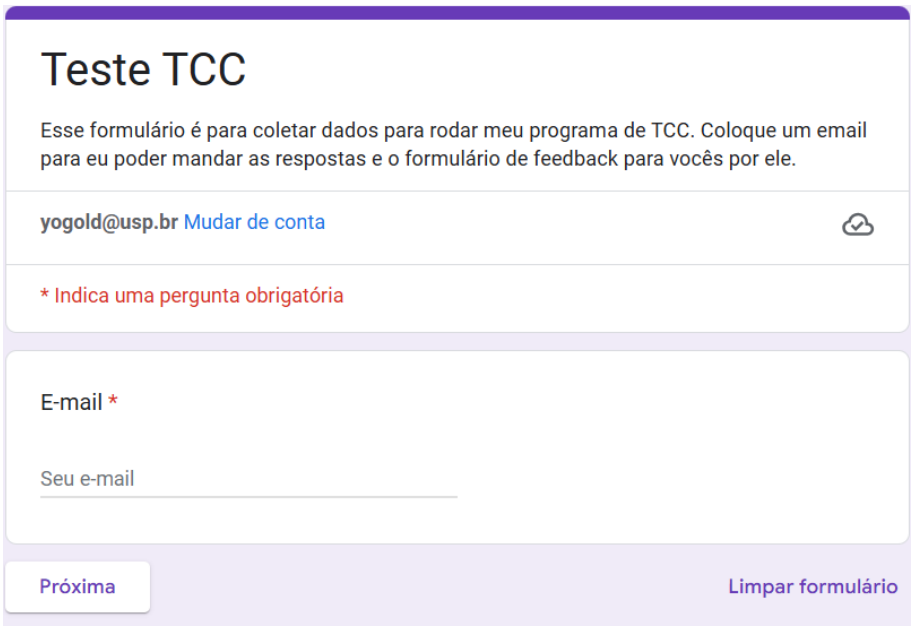
```
#include "User.h"

User::User(string usr, vector<string> games, vector<string> wish, vector<string> gen,
vector<string> cons){
        username = usr;
        ownedGames = games;
        ownedGamesSize = games.size();
        wishlist = wish;
        wishlistSize = wish.size();
        preferedGenres = gen;
        ownedConsoles = cons;
}


string User::getUsername(){return username;}
vector<string> User::getOwnedGames(){return ownedGames;}
vector<string> User::getWishlist(){return wishlist;}
vector<string> User::getPreferedGenres(){return preferedGenres;}
vector<string> User::getOwnedConsoles(){return ownedConsoles;}
int User::getOwnedGamesSize(){return ownedGamesSize;}
int User::getWishlistSize(){return wishlistSize;}
```

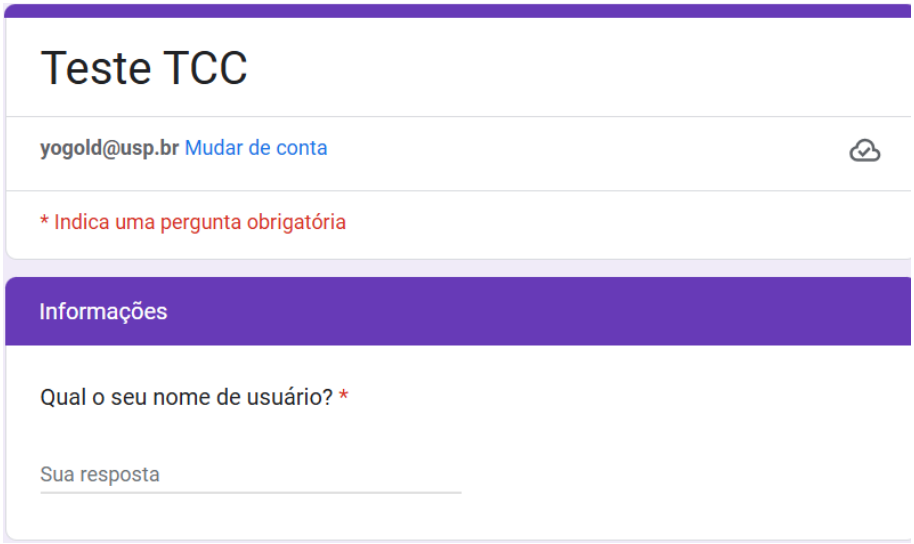
A.5 GameFilter.cpp

```
#include <vector>
#include <string>
#include "GameFilter.h"

bool filterByUser(User pessoa, vector<Jogo> lista){

}

bool filterByDeveloper(string developer, Jogo parsedGame){
    vector<string> devs = parsedGame.getDevelopers();

    if (find(devs.begin(),devs.end(), developer) != devs.end()){
        return true;
    }
    else{
        return false;
    }
}
```

ANEXO B – Formulários

B.1 Formulário de Teste do Protótipo





Quais hardwares de jogos você possui? *

- [ ] Computador Windows
- [ ] Computador Mac
- [ ] Computador Linux
- [ ] Celular Android
- [ ] Celular Apple
- [ ] PS5
- [ ] PS4
- [ ] PS3
- [ ] PS2
- [ ] PS1
- [ ] Nintendo Switch
- [ ] Wii U
- [ ] Wii
- [ ] GameCube
- [ ] N64
- [ ] SNES
- [ ] NES

- [ ] Xbox Series S ou X
- [ ] Xbox One
- [ ] Xbox 360
- [ ] Xbox original
- [ ] 3DS
- [ ] NDS
- [ ] GBA
- [ ] GameBoy
- [ ] PS Vita
- [ ] PSP

Quais jogos você possui? (coloque até 5) *

Sua resposta

Coloque aqui alguns jogos que você deseja: (Até 5) *

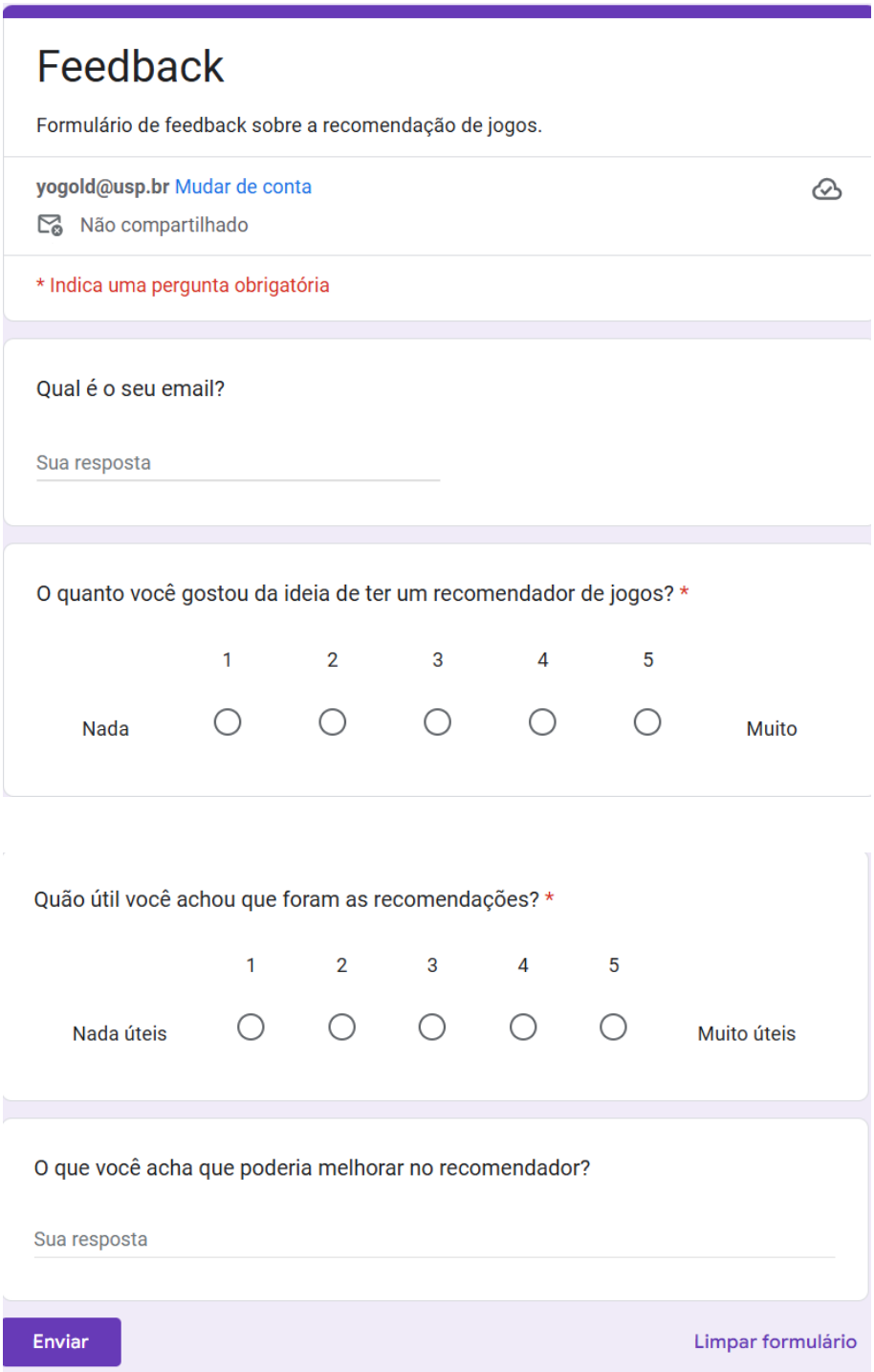
Sua resposta

Que gêneros seus preferidos? *

- [ ] Cartas e Tabuleiro
- [ ] Tático
- [ ] Novela Visual
- [ ] Estratégia de Turno
- [ ] Musica
- [ ] Simulação de vida
- [ ] RPG
- [ ] Esportes
- [ ] Corrida
- [ ] Fliperama
- [ ] Indie
- [ ] Aventura
- [ ] Quebra-cabeça
- [ ] Beat 'em Up
- [ ] Luta
- [ ] Tiro
- [ ] Estratégia Tempo Real
- [ ] Plataforma

Voltar Enviar Limpar formulário

B.2 Formulário de Feedback



Feedback

Formulário de feedback sobre a recomendação de jogos.

yogold@usp.br Mudar de conta

Não compartilhado

* Indica uma pergunta obrigatória

Qual é o seu email?

Sua resposta

O quanto você gostou da ideia de ter um recomendador de jogos? *

	1	2	3	4	5	
Nada	○	○	○	○	○	Muito

Quão útil você achou que foram as recomendações? *

	1	2	3	4	5	
Nada úteis	○	○	○	○	○	Muito úteis

O que você acha que poderia melhorar no recomendador?

Sua resposta

Enviar

Limpar formulário