UNIVERSIDADE FEDERAL DO ABC
Engenharia de informação

Matheus Francisco de Campos dos Santos

Análise comparativa de sistemas de recomendação

Santo André

2019

MATHEUS FRANCISCO DE CAMPOS DOS SANTOS

Análise comparativa de sistemas de recomendação

Trabalho de graduação, realizado sob orientação do Prof. Murilo Bellezoni Loiola, apresentado à Universidade Federal do ABC como requisito para obtenção do título de Bacharel em engenharia de informação.

Santo André
2019

MATHEUS FRANCISCO DE CAMPOS DOS SANTOS

Análise comparativa de sistemas de recomendação

Trabalho de graduação – Universidade federal do ABC

Banca examinadora

_____

Prof. Murilo Bellezoni Loiola

_____

Prof. Aline de Oliveira Neves Panazio

_____

Prof. Ricardo Suyama

Santo André
2019

RESUMO

Com o aumento no volume de dados na internet, os sistemas de recomendação estão, cada vez mais, sendo importantes no auxílio ao usuário na hora de encontrar o item desejado. A crescente facilidade de acesso a serviços de streaming de músicas, como o Spotify, que possuem grandes bibliotecas musicais, torna o estudo de recomendações nessa área importante.

Este trabalho fornece uma visão geral dos tipos de sistemas de recomendação existentes, e a partir dos dados disponibilizados pela Last FM, implementa diferentes tipos de sistemas de recomendação, com técnicas de filtragem colaborativa e filtragem baseada em conteúdo, comparando qual técnica gera melhores resultados ao usuário.

Palavras-chave: Sistemas de recomendação, recomendação de música, filtragem colaborativa, avaliação de sistema de recomendação.

ABSTRACT

Due to increased volume of data on the Internet, recommender systems become increasingly important in helping users to find the desired itens. The increasing ease of access to streaming music services, such as Spotify, which have large music libraries, makes recommendations study important in this area.

This work provides an overview of existing types of recommender systems and, based on the data provided by Last FM, implements different types of recommender systems with techniques of collaborative filtering and content based filtering, comparing which technique generates better results for the user.

Keywords: Recommender systems, music recommendation, collaborative filtering, recommender systems evaluation.

LISTA DE EQUAÇÕES

Equação 1- Semelhança do cosseno 12
Equação 2- Correlação de Pearson 13
Equação 3- Precisão 21
Equação 4 - Average Precision. 22
Equação 5 - Mean Average Precision 22
Equação 6 - Revocação 22
Equação 7 - Cobertura 23
Equação 8 - Entropia 23
Equação 9 - Semelhança do cosseno Adaptada 31

LISTA DE FIGURAS

Figura 1 - Principais etapas e métodos em um problema de mineração de dados. 11
Figura 2- a – Exemplo do algoritmo KNN.....15
Figura 3- JSON para o método getTopTracks.....26
Figura 4- Diagrama de banco de dados.....29

SUMÁRIO

1. INTRODUÇÃO 10

1.1. Motivação 10

1.2. Objetivo 11

2. MÉTODOS DE MINEIRAÇÃO DE DADOS EM SISTEMAS DE RECOMENDAÇÃO ..11

2.1. Processamento de dados 12

2.1.1. Similaridade 12

2.1.2. Amostragem 13

2.1.3. Dimensionamento 13

2.2. Analise dos dados 14

3. SISTEMAS DE RECOMENDAÇÃO 15

3.1. Filtragem Colaborativa 17

3.2. Filtragem baseada em conteúdo 19

3.3. Sistemas Híbridos 20

4. AVALIAÇÃO DE SISTEMAS DE RECOMENDAÇÃO 21

4.1. Precisão 21

4.2. Precision at M 21

4.3. Average Precision(AveP) 22

4.4. Mean Average Precision(MAP) 22

4.5. Revocação(Recall) 22

4.6. Cobertura 23

5. METODOLOGIA 24

5.1. BASE DE DADOS 24

5.1.1. Banco de Dados 24

5.1.2. Obtenção dos Dados 24

5.1.3. Metadados 27

5.1.4. Preparação dos Dados 29

5.2. RECOMENDAÇÃO DE MÚSICAS 30

5.2.1. Filtragem Colaborativa 30

5.2.2. Filtragem Baseada em Conteúdo 31

5.2.3. Sistemas Hibridos 32

6. RESULTADOS 34

7. CONCLUSÃO 36

8. REFERÊNCIAS BIBLIOGRÁFICAS 37

9. APÊNDICE A – Classe SQL, conexão e query 39
10. APÊNDICE B – Classe topTracks 41
11. APÊNDICE C – Classe colaborative 43
12. APÊNDICE D – Classe content 46

1. INTRODUÇÃO

1.1. Motivação

Com o aumento expressivo da quantidade de dados gerados pela internet e pelas empresas nos últimos anos, junto com os avanços tecnológicos que possibilitam o armazenamento e processamento dessa grande quantidade de dados, foram criadas diversas oportunidades que possibilitam uma tomada de decisão mais assertiva com base na análise desses dados, gerando bons resultados à quem os utiliza para tomada de decisão.

Porém, o grande volume de dados gerados pode prejudicar os usuários na tarefa de recuperar o conteúdo que interessa. Os sistemas de recomendação surgiram para auxiliar o usuário a filtrar apenas o que mais lhe interessa.

O objetivo dos sistemas de recomendação é gerar recomendações dos itens mais interessantes ao usuário, gerando predições baseadas nas preferências e restrições do usuário.

Para completar tal tarefa computacional, o sistema de recomendação coleta dos usuários suas preferências, que são explicitamente expressas, como classificações de produtos, ou são inferidas por interpretação de ações do usuário. Podemos ter diversos tipos de recomendações. Sugestões de produtos em lojas online, filmes e séries em aplicativos são exemplos de itens que o sistema pode recomendar ao usuário. Filtrando os itens mais interessantes ao usuário podemos ter diversos benefícios como manter o usuário utilizando o nosso produto ou tomar a decisão mais assertiva para a empresa.

Os sistemas de recomendação são geralmente classificados em filtragem colaborativa, filtragem baseada em conteúdo e sistemas híbridos. Em geral, sistema de recomendação que utiliza filtragem colaborativa usa uma técnica de filtragem de informações com base na avaliação anterior do usuário de itens ou histórico de compras anteriores, e recomenda produtos com boas avaliações de outros usuários com histórico parecido. O sistema de recomendação que utiliza filtragem baseada em conteúdo analisa um conjunto de documentos classificados por um usuário individual e usa informações sobre os documentos, bem como as classificações fornecidas, para inferir um perfil de usuário, gerando recomendações de acordo com o perfil inferido. Sistema de recomendação híbrido combina técnicas dos dois outros tipos [3].

Existem outros estudos de recomendação de músicas que utilizam o sinal do áudio das músicas para realizar as recomendações. Este método, porém, sempre apresenta piores resultados quando comparados com os que sistemas de recomendação que utilizam filtragem baseada em conteúdo, devido à dificuldade de relacionar músicas apenas pelo conteúdo do áudio [12]. Vários

estudos mostram que a filtragem colaborativa apresenta os melhores resultados de recomendação de músicas [12].

1.2. Objetivo

Este trabalho de graduação tem como objetivo desenvolver diferentes tipos de sistemas de recomendação para músicas, aplicados com base nos dados de usuários adquiridos a partir do API da Last FM, e realizar comparações entre os sistemas que utilizam filtragem colaborativa, filtragem baseada em conteúdo e sistemas híbridos.

2. MÉTODOS DE MINEIRAÇÃO DE DADOS EM SISTEMAS DE RECOMENDAÇÃO

O processo de mineração de dados normalmente consiste em 3 etapas, executadas em sucessão: Pré-processamento de dados, análise de dados e interpretação de resultados. A figura 1 detalha diferentes tópicos presentes em cada etapa da mineiração de dados.

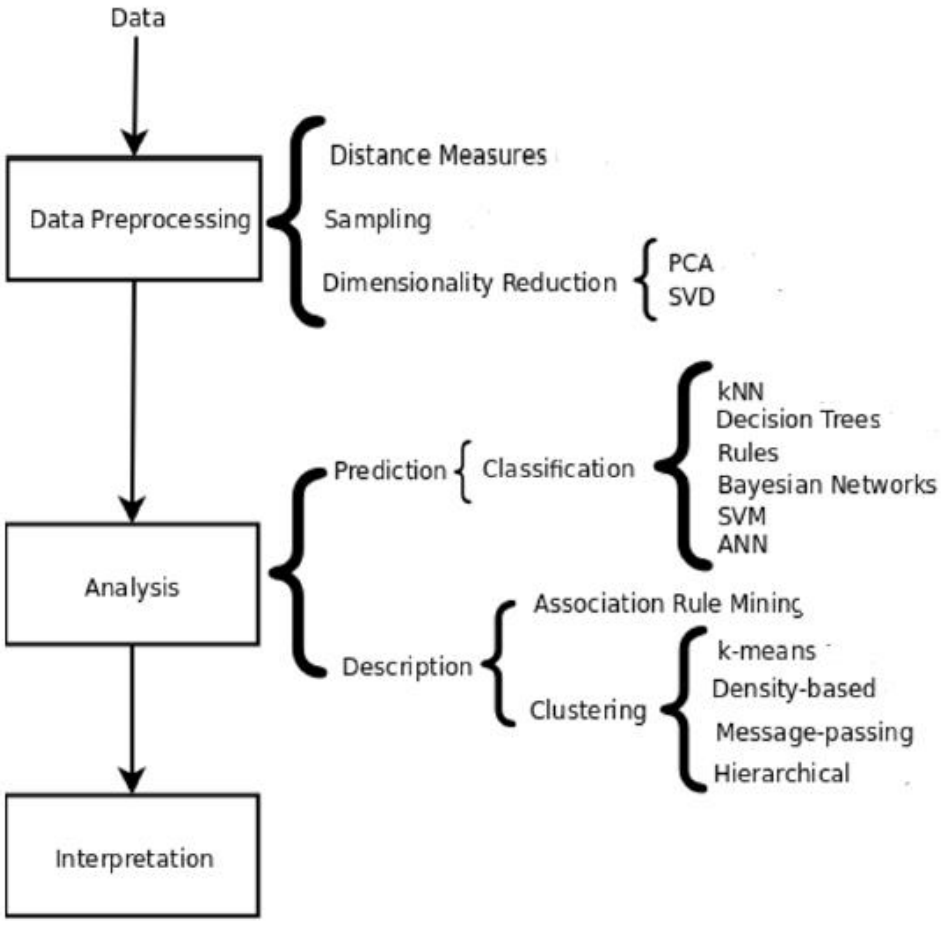

Figura 1 - Principais etapas e métodos em um problema de mineração de dados
Fonte: Recommender Systems handbook [1]

Na sequencia detalharemos algumas das técnicas mostradas na figura 1, detalhando melhor aquelas que foram utilizadas neste trabalho .

2.1. Processamento de dados

Nós definimos os dados como uma coleção de objetos e seus atributos, onde um atributo é definido como uma propriedade ou característica de um objeto. Outros nomes para objeto incluem registro, item, ponto, amostra, observação ou instância. Um atributo também pode ser referido como uma variável, campo, característica ou recurso [1].

Os dados normalmente precisam ser pré-processados (por exemplo, filtrados ou transformados) para serem utilizados pelas técnicas de aprendizado de máquina na etapa de análise. Para sistemas de recomendação, três questões que são de particularmente importantes: medidas de similaridade, amostragem e dimensionamento. Estas definições serão descritas a seguir.

2.1.1. Similaridade

O conceito de semelhança é fundamental para sistemas de recomendação que utilizam vizinhança como método de classificação, sendo que a semelhança pode ser obtida de diversas maneiras [3]. A semelhança de cosseno e o coeficiente de correlação de Pearson são duas das métricas mais utilizadas para se obter a semelhança entre usuários e itens.

A semelhança de cosseno entre dois vetores calcula o ângulo entre eles, sendo que o cosseno do ângulo pode estar entre 0 e 1. Quanto maior o cosseno entre os vetores, mais semelhantes eles são [3]. Em termos geométricos, a semelhança de cosseno está relacionada à projeção de um vetor sobre o outro.

A semelhança de cosseno é obtida pela equação 1:

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Equação 1- Semelhança do cosseno

onde Θ representa o ângulo entre os vetores A e B.

Já o coeficiente de correlação de Pearson mede o grau de associação entre duas variáveis, sendo que esse coeficiente pode variar entre 1 e -1. [3]

Quando o coeficiente de correlação (r) é igual a 0, significa que as duas variáveis não dependem linearmente uma da outra, quando r é igual a 1, significa uma correlação perfeita entre as duas variáveis, ou seja, quando uma aumenta, a outra também aumenta e vice versa, e, por fim, quando r é igual a -1, significa uma correlação negativa perfeita entre as duas variáveis.

O coeficiente de correlação de Pearson pode ser obtido a partir da equação 2:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\left[\sum_{i=1}^{n}(x_i - \bar{x})^2\right]\left[\sum_{i=1}^{n}(y_i - \bar{y})^2\right]}}$$

Equação 2- Correlação de Pearson

O r é o coeficiente de Correlação de Pearson medido para os vetores x e y, sendo que xi e yi, são elementos dos vetores x e y, respectivamente, e $\bar{x}$ e $\bar{y}$ são os valores médios de seus respectivos vetores.

Existem ainda outras formas de se obter a semelhança como: Distância Euclidiana, diferenças quadradas médias e correlação restrita [5]. Neste trabalho, a medida de semelhança utilizada foi a de semelhança de cosseno.

2.1.2. Amostragem

A amostragem é usada para selecionar um subconjunto de dados relevantes de um grande conjunto de dados. É usado tanto na etapa de pré-processamento quanto na etapa de interpretação final dos dados. A amostragem é usada porque o processamento de todos os dados geralmente é computacionalmente muito custoso. Também pode ser usada para criar conjuntos de dados de treinamento e testes. Neste caso, o conjunto de dados de treinamento é usado para aprender os parâmetros ou configurar os algoritmos usados na etapa de análise, enquanto o conjunto de dados de teste é usado para avaliar o modelo ou configuração obtida na fase de treinamento, certificando-se de que ele funcione bem com dados inéditos.

O principal objetivo da amostragem é, portanto, encontrar um subconjunto do conjunto de dados original que seja representativo, isto é, tem aproximadamente a mesma propriedade de interesse de todo o conjunto. A técnica de amostragem mais simples é a amostragem aleatória, onde existe uma probabilidade igual de selecionar qualquer item [1], utilizada no trabalho para separar a base de teste.

2.1.3. Dimensionamento

É comum um sistema de recomendação possuir não apenas um conjunto de dados com recursos que definem um espaço, mas também informações muito esparsas nesse espaço, ou seja, valores para um número limitado de recursos por objeto. As noções de densidade e distância entre pontos, que são críticos para agrupamentos e detecção de outliers, tornam-se menos significativos em espaços altamente dimensionais. As técnicas de redução de dimensionalidade ajudam a superar esse problema transformando o espaço original de alta dimensão em uma dimensionalidade menor.

A esparsidade é um problema recorrente em sistemas de recomendação. Mesmo na configuração mais simples, é provável que tenhamos uma matriz esparsa com milhares de linhas e colunas (ou seja, usuários e itens), a maioria dos quais são zeros. Principal *Component Analysis* (PCA) [12] e *Singular Value Decomposition* (SVD) são exemplos de algoritmos aplicados no dimensionamento da base de dados [1], tais técnicas não foram utilizadas no trabalho.

2.2. Analise dos dados

Na etapa de analise ocorre o processamento dos dados obtidos. Existem diversos métodos para realizar a analise de dados. Para sistemas de recomendação, os métodos de classificação e de agrupamento são mais utilizados [1].

Métodos de agrupamento dividem um conjunto de itens em grupos, sendo que itens dos mesmos grupos possuem alguma semelhança que fazem com que eles sejam agrupados naquele grupo. *K-means* é um exemplo de método de agrupamento [1], no entanto neste trabalho, focamos no método de classificação.

Métodos de classificação são responsáveis por classificar itens em diferentes classes. Um classificador de e-mail, por exemplo, pode classificar um e-mail como normal ou *span*. Um método de classificação que será apresentado a seguir é o KNN (*K-Nearest Neighbors*) [1].

Para gerar recomendações a partir de dados dos outros usuários, o algoritmo KNN é separado em três passos: primeiramente, ele determina o número (K) de usuários vizinhos ao usuário ativo (a), depois implementa uma classificação a partir dos itens avaliados pelos vizinhos e não avaliado pelo usuário ativo e, por fim, ele gera M recomendações com base nos itens com melhores classificações.

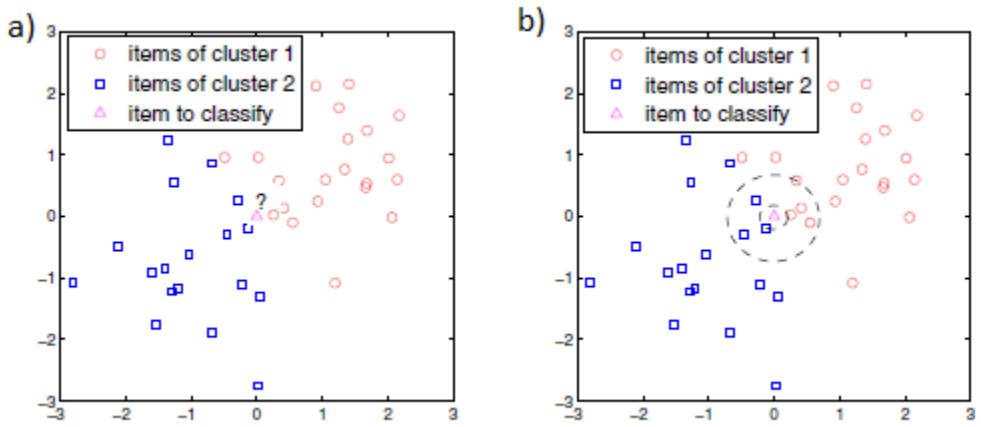

Figura 2- a – Exemplo do algoritmo KNN

Figura 2- b – Algoritmo KNN para diferentes valores de K

Fonte: Recommender Systems Handbook [1]

Na figura 2-a, enxergamos dois tipos de classes (círculos e quadrados), sendo que o algoritmo deseja classificar o item do centro (triangulo) em uma das duas classes. Se o algoritmo considerar K=1, o item seria classificado como um quadrado. Caso o algoritmo considere K= 7, o item seria classificado como um círculo, uma vez que a maioria dos itens considerados entre os vizinhos mais próximos são círculos, conforme ilustrado na figura 2-b [1].

O maior desafio do algoritmo KNN é determinar o valor de K. Caso ele seja muito baixo, o classificador pode ficar muito sensível. Contudo, caso ele seja muito alto, o classificador pode considerar muitas classes e acabar perdendo a sensibilidade em relação ao item que se deseja classificar [1].

3. SISTEMAS DE RECOMENDAÇÃO

Um sistema de recomendação é um conjunto de algoritmos e técnicas que realizam sugestões de itens a serem usados por um usuário. Em geral, um sistema de recomendação nos ajuda a escolher sites, músicas, programas de TV, livros entre outros produtos e serviços, a partir de comparações e estudos analíticos realizados com base no perfil do usuário, que permitem calcular a probabilidade de um usuário adquirir ou gostar de um produto em questão. O perfil do usuário é gerado a partir dos dados que temos disponíveis sobre ele. Os dados podem ser obtidos de forma implícita, como o histórico de navegação do usuário, ou de forma explícita, como notas dadas pelo usuário a um produto.

Existem várias razões para que os provedores de serviços utilizem sistemas de recomendação:

- **Aumentar o número de itens vendidos**. Este objetivo é alcançado quando o item recomendado atende as necessidades e desejos do usuário. Podemos presumir que o usuário reconhecerá isso depois de ter tentado diversas recomendações. O principal objetivo do provedor de serviço ao introduzir um sistema de recomendação é aumentar a taxa de conversão, ou seja, aumentar o numero de usuários que aceitam a recomendação e consomem o produto em relação aos usuários que apenas navegam pelas informações [1].

- **Vender itens mais diversos**. Outra função importante de um sistema de recomendação é habilitar o usuário para selecionar itens que podem ser difíceis de encontrar sem uma recomendação precisa. Por exemplo, em um fornecedor de filme por streaming, o provedor de serviços está interessado que todos os filmes do catálogo sejam vistos, não apenas os mais populares. Isto poderia ser difícil sem um sistema de recomendação, pois o provedor de serviços não pode arcar com o risco de filmes publicitários que provavelmente não atendem ao gosto de um usuário em particular. Assim sendo, um sistema de recomendação sugere ou anuncia filmes impopulares para os usuários certos [1].

- **Aumentar a satisfação do usuário.** Um sistema de recomendação bem projetado também pode melhorar a experiência do usuário com o site ou o aplicativo. O usuário encontrará as recomendações interessantes e relevantes, e com uma interface amigável, o usuário também vai gostar de interagir com o sistema. A combinação de eficácia, ou seja, recomendações precisas e uma interface agradável aumentarão o número de usuários e a utilização da avaliação dos itens. Isso, por sua vez, aumentará o uso do sistema e a probabilidade de que as recomendações sejam aceitas [1].

- **Aumentar a fidelidade do usuário**. Um usuário deve ser fiel a um site que, quando visitado, reconhece o antigo cliente e o trata como um visitante valioso. Isso é uma característica normal de um sistema de recomendação, já que muitos sistemas de recomendação computam recomendações, levantando informações adquiridas do usuário em interações anteriores. Quanto mais tempo o usuário interage com o site,

mais refinado seu modelo de usuário se torna, gerando recomendações mais assertivas [1].

Sistemas de recomendação são sistemas de processamento de informações que coletam ativamente vários tipos de dados a fim de construir suas recomendações. Os dados são principalmente sobre os itens a sugerir e os usuários que receberão essas recomendações. Mas, sabendo que os dados e as fontes de conhecimento disponíveis para os sistemas de recomendação podem ser muito diversas, em última análise, se eles podem ser explorados ou não depende da técnica de recomendação.

Em geral, existem técnicas de recomendação que são mais simples, ou seja, utilizam dados muito básicos, como classificações ou avaliações de usuários sobre os itens. Outras técnicas são muito mais complexas, exigindo por exemplo, descrições detalhadas dos usuários ou dos itens, ou restrições, ou relações sociais e atividades dos usuários. De qualquer maneira os dados utilizados pelos sistemas de recomendação referem-se a três tipos de objetos: itens, usuários e transações, ou seja, relações entre usuários e itens [1].

Como comentado na seção 1, sistemas de recomendação são geralmente classificados entre três categorias sendo elas: Filtragem colaborativa (*collaborative filtering*), filtragem baseada em conteúdo (*content-based filtering*) e sistemas híbridos (*Hybrid Recommender Systems*).

Abaixo será explicado em detalhes o funcionamento de cada tipo de sistema de recomendação.’

3.1. Filtragem Colaborativa

Filtragem colaborativa leva em consideração o que um usuário com gosto semelhante prefere, para gerar a recomendação de um item ao usuário final. Sistemas de recomendação que utilizam filtragem colaborativa podem ser divididos em duas grandes classes: métodos baseados em vizinhança e métodos baseados em modelo [2].

Para métodos baseados em modelo, a ideia geral é modelar as interações item-usuário com fatores que representam características relevantes dos usuários e itens no sistema. Este modelo é então treinado usando os dados disponíveis e, posteriormente, usados para prever classificações de usuários para novos itens. Existem diversos modelos que podem ser utilizados para tal finalidade, *Bayesian Clustering*, *Latent Semantic Analysis* e *Boltzmann Machines* são exemplos de modelos [1].

Entre os métodos baseados em vizinhança, o algoritmo mais utilizado para filtragem colaborativa é o *k Nearest Neighbors* (kNN) [2]. Um sistema de recomendação que utiliza filtragem

colaborativa pode recomendar, a partir de dados de usuários que tenham gostos parecidos, músicas desconhecidas ao usuário final, que são populares entre os usuários com gosto semelhante. Uma desvantagem dessa categoria de algoritmos é que ela precisa de uma grande quantidade de informações sobre usuários para gerar as recomendações.

As maiores vantagens de se usar modelos baseados em vizinhança na filtragem colaborativa são:

- **Simplicidade**. Os métodos baseados em filtragem colaborativa são intuitivos e são simples de serem implementados. Na sua forma mais simples, apenas um parâmetro (o número de vizinhos usado na previsão) é utilizado.

- **Eficiência.** Um dos pontos fortes da filtragem colaborativa é sua eficiência. Ao contrário da maioria dos sistemas baseados em modelos, eles não exigem fases de treinamento. Embora a fase de recomendação seja geralmente mais demorada do que métodos baseados em modelo, os vizinhos mais próximos podem ser pré-computados, fornecendo recomendações quase instantâneas. Além disso, armazenar esses vizinhos mais próximos requer pouca memória, tornando essas abordagens escaláveis para aplicativos com milhões de usuários e itens.

- **Estabilidade**. Outra vantagem dos sistemas de recomendação que utilizam essa abordagem é que eles são pouco afetados pela adição constante de usuários, itens e classificações. Por exemplo, uma vez que as similaridades dos itens tenham sido computadas, um sistema baseado em itens pode realizar prontamente recomendações para novos usuários, sem precisar treinar novamente o sistema.

Contudo, a filtragem colaborativa possui duas falhas importantes [1]:

- **Cobertura limitada**. A similaridade entre dois usuários é medida pela similaridade entre suas classificações, comparando suas avaliações para os mesmos itens. Os usuários podem ser vizinhos somente se eles tiverem classificado itens comuns. Essa suposição é muito limitante, já que os usuários que avaliaram alguns ou nenhum item comum, ainda podem ter preferências semelhantes. Além disso, já que apenas itens classificados por vizinhos podem ser recomendados, a quantidade de itens recomendados por filtragem colaborativa pode ser limitada.

- **Sensibilidade a dados esparsos.** A precisão de sistemas que utilizam a filtragem colaborativa diminui com a falta de classificações disponíveis. A grande quantidade de itens no catálogo é um problema comum à maioria dos sistemas de recomendação devido ao fato de que os usuários normalmente classificam apenas uma pequena proporção dos itens disponíveis. Este problema é agravado pelo fato de que usuários ou itens recém-adicionados ao sistema podem ter nenhuma avaliação, um problema conhecido como *cold-start* [1].

3.2. Filtragem baseada em conteúdo

Esta categoria de sistema de recomendação recomenda ao usuário itens semelhantes àqueles que o usuário interagiu no passado. Caso o usuário tenha assistido algum filme de ficção no passado e o classificou de forma positiva, o sistema irá recomendar novos filmes de ficção que o usuário ainda não assistiu. Os algoritmos que realizam a recomendação baseada em conteúdo consideram interesses do usuário e relacionam aos atributos dos itens disponíveis para recomendar ao usuário novos itens [1].

A utilização da filtragem baseada em conteúdo possui algumas vantagens quando comparada à filtragem colaborativa:

- **Independência do usuário**. Sistemas de recomendação baseados em conteúdo exploram apenas classificações fornecida pelo usuário ativo para criar seu próprio perfil, enquanto, a filtragem colaborativa precisa de classificações de outros usuários para encontrar os "vizinhos mais próximos" do usuário ativo, ou seja, usuários que têm gostos semelhantes.

- **Transparência**. Explicações sobre como funciona o sistema de recomendação podem ser fornecidas explicitamente listando os recursos do sistema e descrevendo características que fizeram um item ser recomendado.

- **Novo item**. Recomendadores baseados em conteúdo são capazes de recomendar itens ainda não classificado por nenhum usuário. Como conseqüência, eles não sofrem com o problema do primeiro avaliador, que afeta os sistemas que utilizam filtragem colaborativa, que dependem exclusivamente dos usuários para fazer recomendações. Portanto, até que o novo item seja avaliado por um número significativo de usuários, o sistema não será capaz de recomendá-lo.

A filtragem baseada em conteúdo possui, porém, algumas desvantagens:

- **Análise de conteúdo limitada.** Técnicas de recomendação baseadas em conteúdo têm um limite natural no número e tipo de recursos que estão associados com os objetos que eles recomendam. O conhecimento do domínio é necessário, por exemplo, para recomendações de filmes. O sistema precisa conhecer os atores, diretores, gênero, entre outras informações sobre o seu catálogo de filmes.

- **Especialização.** Recomendadores baseados em conteúdo não possuem nenhum método para encontrar algo inesperado. O sistema sugere itens cujas pontuações são altas quando comparado com o perfil do usuário. Portanto, ao usuário serão recomendados itens semelhantes aos que já foram classificados.

- **Novo usuário.** Classificações suficientes devem ser coletadas antes que um sistema de recomendação baseado em conteúdo possa realmente entender as preferências do usuário e fornecer informações precisas de recomendações. Portanto, quando poucas classificações estão disponíveis, o sistema não poderá fornecer recomendações confiáveis

O tipo de perfil do usuário gerado pelo sistema de recomendação baseado em conteúdo depende do método de aprendizagem que está sendo aplicado no sistema. Arvores de decisão e redes neurais são exemplos de métodos que podem ser aplicados a esse sistema [4].

3.3. Sistemas Híbridos

Um sistema de recomendação híbrido combina duas ou mais técnicas de recomendação para melhorar o desempenho do sistema. Duas técnicas de recomendação podem ser combinadas de diversas formas. Sistemas de recomendação podem ser combinados por pesos, determinando diferentes pesos para as diferentes técnicas de recomendação presentes no sistema, recomendando os itens que obtiverem a maior pontuação [4].

Outra forma de se combinar duas técnicas de recomendação em um sistema hibrido é por troca, onde um critério é adotado para se determinar qual técnica de recomendação será utilizada para o usuário em questão [4].

4. AVALIAÇÃO DE SISTEMAS DE RECOMENDAÇÃO

Pesquisas em sistemas de recomendação precisam de métricas de avaliação para saber a qualidade das técnicas, métodos e algoritmos utilizados na recomendação. Métricas de avaliação facilitam a comparação entre diferentes métodos utilizados para a recomendação, determinando qual gera melhores resultados.

Como o objetivo do sistema de recomendação é recomendar apenas itens classificados como interessantes ao usuário, quando o sistema recomenda itens classificados como não interessantes ao usuário ele erra. Na tabela abaixo, denominada como matriz de confusão para uma classificação binária, podemos ver os resultados possíveis dessa classificação.

Tabela 1- Matriz de confusão para uma classificação binária.

	Recomendado	Não Recomendado
Interessante	Verdadeiro - Positivo (VP)	Falso - Negativo(FN)
Não Interessante	Falso - Positivo (FP)	Verdadeiro - Negativo (VN)

Utilizando a matriz de confusão é possível obter fórmulas comumente usadas para avaliação de sistemas de recomendação. No trabalho foram utilizadas apenas as fórmulas de precisão e revocação para realizar a avaliação dos sistemas, devido aos dados disponíveis para análise.

4.1. Precisão

A precisão do sistema é dada pela fração entre os itens recomendados que são de interesse do usuário e todos os itens recomendados [13].

$$Precisão = \frac{VP}{VP + FP}$$

Equação 3- Precisão

4.2. Precision at M

Em aplicações onde o número de recomendações (M) é predeterminado podemos utilizar a medida de avaliação conhecida como Precision at M [13].

Neste caso, considera-se somente os M primeiros itens da recomendação e quando nenhum item de interesse é recuperado, a precisão do sistema é zero.

Para utilizar esse método de avaliação são necessárias diversas simulações para diferentes M.

4.3. Average Precision(AveP)

Em um sistema de recomendação do tipo top M, que recomenda M itens mais interessantes, é valido considerar a posição dos itens relevantes na lista e uma medida de precisão para isso é a Average Precision (AveP) [13].

$$AveP = \frac{\sum_{M=1}^{n}(P(k)xrel(M))}{R}$$

Equação 4 - Average Precision

onde n é a quantidade de itens recuperados na lista de recomendação e R é a quantidade de documentos relevantes, neste caso a quantidade de itens removidos do perfil do usuário da base de treino e movidos para a base de validação. Nele M é a posição do documento no rank (catálogo) e a função P(M) é a precisão, calculada pela equação 3, no ponto de corte M da lista. A função rel(M) indica valor 1 se o item nesta posição M do somatório é interessante ao usuário, e indica valor 0, em caso contrário.

4.4. Mean Average Precision(MAP)

Essa medida é o resumo, de valor único, de todas as recomendações, sendo dada por:

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

Equação 5 - Mean Average Precision

onde Q é a quantidade de itens recomendados pelo sistema.

4.5. Revocação(Recall)

É dada pela fração entre a quantidade de itens recomendados que são do interesse do usuário e todos os itens de interesse do usuário. É dada por:

$$Recall = \frac{VP}{VP + FN}$$

Equação 6 - Revocação

Um trade off entre medidas de Precisão e Revocação é esperado na avaliação de sistemas de recomendação. Aumentar o número de recomendações pode melhorar a Revocação do sistema, porém é esperado uma queda na Precisão.

Um sistema de recomendação aplicado a vários usuários, onde é recomendado um número K de itens para cada usuário, pode ser avaliado traçando um gráfico equivalente à curva ROC (Receiver Operating Characteristic), que é uma forma eficiente de demonstrar a relação, normalmente antagônica, entre Precisão e Revocação[13].

4.6. Cobertura

O termo cobertura refere-se à proporção de itens que um sistema de recomendação pode recomendar. Muitas vezes é chamado de cobertura do catálogo.

A medida mais simples de se determinar a cobertura do catálogo é obtendo o percentual entre todos os itens presentes na lista de recomendação (M) e todos os itens do catálogo (Nc)[13].

$$Cobertura = \frac{M}{Nc} x100$$

Equação 7 - Cobertura

Existem diversas formas de se medir a cobertura do catálogo em um sistema de recomendação. Uma medida que considera a desigualdade distributiva das recomendações é a entropia de Shannon[6].

$$H = -\sum_{i=1}^{n} p(i) \log p(i)$$

Equação 8 - Entropia

onde H é a entropia, p(i) é a probabilidade de recomendar o item e n é o número de itens no catálogo.

A entropia é 0 quando um único item é sempre escolhido, e log n quando n itens são escolhidos igualmente. Os valores 0 e log n são, respectivamente, os limitantes inferior e superior da entropia.

5. METODOLOGIA

5.1. BASE DE DADOS

5.1.1. Banco de Dados

Como SGBD (sistema gerenciador de banco de dados) foi utilizado o SQL Server da Microsoft. O SQL Server é um mecanismo de banco de dados para armazenamento, processamento e segurança de dados. O mecanismo de banco de dados fornece acesso controlado e processamento rápido de transações para atender as mais diversas aplicações. O SQL Server permite a criação de tabelas relacionais e objetos de banco de dados, como índices e procedimentos armazenados, para exibição, gerenciamento e segurança dos dados. No trabalho o SQL Server foi utilizado para armazenamento dos dados a partir de tabelas [9].

Para configurar e administrar os dados gravados no SQL Server, foi utilizado outro software da Microsoft, chamado SQL Server Management Studio. O SQL Server Management Studio é um software integrado ao SQL Server que permite gerenciar a sua estrutura. O Management Studio fornece ferramentas para configurar, monitorar e administrar as instancias do SQL Server, fornecendo, também, ferramentas para implantar, monitorar e atualizar os componentes da camada de dados através de uma interface amigável. No trabalho o Management Studio foi utilizado para realizar consultas na linguagem T-SQL (Transact-SQL), visualizar os dados e realizar o gerenciamento das tabelas [10].

5.1.2. Obtenção dos Dados

Para criar a base de dados com informações de usuários e músicas foi utilizado a API (*aplication programming interface*) da Last FM. A Last FM é um serviço musical que aprende seu gosto musical. O aplicativo da Last FM armazena o que o usuário ouve, processo chamado de *scrobbling*, e a partir das músicas escutadas pelo usuário o aplicativo gera recomendações de novas músicas, gera seu próprio perfil musical e exibe todo seu histórico musical, permitindo que o usuário veja, por exemplo, quando foi a primeira vez que ouviu uma música específica [11].

O API da Last FM permite a aquisição de dados dos usuários que permitem tal compartilhamento, e de sua biblioteca de músicas a partir de métodos que retornam páginas com os dados requisitados no formato JSON. JSON (*JavaScript Object Notation*) é um modelo para armazenamento e transmissão de informações no formato texto.

Para utilizar o API da Last FM é necessária uma API *key*, que permite a autenticação da sua aplicação com a API. Uma API key é obtida, gratuitamente, criando um usuário no site da Last FM.

A captura dos dados a partir do API da Last FM foi implementada em linguagem Java, utilizando o programa Eclipse. O programa criado é responsável por realizar a requisição no API e capturar o JSON a partir do comando GET do protocolo HTTP, como o utilizado na classe topTracks [APENDICE B]. O API da Last FM possui diversos métodos parametrizados, que retornam dados específicos.

Utilizando os métodos do API para se obter as informações no formato JSON, foi necessário realizar o *parsing* para se obter as informações de maneira estruturada em formato de tabelas. Para isso foi necessário importar uma biblioteca não nativa do JAVA especifica para JSON chamada JSON-JAVA [7].

A comunicação entre a aplicação na linguagem JAVA e o banco de dados é feita a partir da API JDBC (*Java Database Connectivity*)[8], que permite que a aplicação execute comandos no banco de dados. No trabalho foi criado uma classe chamada SQL [APENDICE A], nela foram criados todos os métodos que realizaram alguma consulta com o banco de dados. O construtor da classe exige quatro argumentos, sendo eles o nome do servidor, o nome do usuário, senha e o banco de dados que se deseja conectar. Dessa forma sempre que for necessário a realização de alguma consulta no banco de dados uma nova instancia da classe SQL deve ser iniciada e o método connect [APENDICE A] deve ser invocado para iniciar a conexão entre o programa JAVA e o servidor SQL.

Para gerar a base de dados do trabalho foram utilizados os métodos:

- **user.getTopTracks**. Retorna as músicas mais escutadas pelo usuário no formato JSON. O método é executado preenchendo a URL: http://ws.audioscrobbler.com/2.0/?method=user.gettoptracks&user=USER&api_key=API_KEY&format=json. Como parâmetros do método temos API_KEY, que é a chave da sua aplicação e USER que é o nome do usuário que se deseja obter as informações. A figura 3 mostra parte do JSON que é retornado quando utilizamos a função user.getTopTracks

```
{
  "toptracks": {
    "@attr": {
      "total": "8636",
      "perPage": "50",
      "totalPages": "173",
      "page": "1",
      "user": "luke_10"
    },
    "track": [
      {
        "duration": "251",
        "image": [
          {
            "#text": "https://lastfm-img2.akamaized.net/i/u/34s/abb1b06b13b64479b66eb4d8e13c9478.png",
            "size": "small"
          }
        ],
        "@attr": {
          "rank": "1"
        },
        "mbid": "11c01f77-0931-486e-b31c-1278b4f4441a",
        "artist": {
          "mbid": "fd857293-5ab8-40de-b29e-55a69d4e4d0f",
          "name": "Muse",
          "url": "https://www.last.fm/music/Muse"
        },
        "playcount": "130",
        "name": "Supermassive Black Hole",
        "url": "https://www.last.fm/music/Muse/_/Supermassive+Black+Hole"
      },
      {
        "duration": "245",
        "image": [
          {
            "#text": "https://lastfm-img2.akamaized.net/i/u/34s/7424dadf00345965e5c783aa6f1ef40d.png",
            "size": "small"
          }
        ],
        "@attr": {
          "rank": "2"
        },
```


Figura 3- JSON para o método getTopTracks

- **user.getFriends** Retorna o nome dos amigos do usuário na Last FM no formato JSON. O método é executado preenchendo a URL: http://ws.audioscrobbler.com/2.0/?method=user.getfriends&user=USER&api_key=API_KEY&format=json. Como parâmetros do método temos API_KEY, que é a chave da sua aplicação e USER que é o nome do usuário sobre o qual se deseja obter as informações.

- **user.getInfo.** Retorna o nome real, idade, gênero e país do usuário além da quantidade de playlists e quantidade de músicas escutadas pelo usuário no formato JSON. O método é executado preenchendo a URL: http://ws.audioscrobbler.com/2.0/?method=user.getinfo&user=USER&api_key=API_KEY&format=json. Como parâmetros do método temos API_KEY e USER.

- **track.getInfo.** Retorna o nome da música, ID da música na aplicação, nome do artista, ID do artista na aplicação, álbum da música, ID do álbum na aplicação, TAGs que são palavras que os usuários utilizam para classificar o gênero da música, além da quantidade de vezes que a música foi escutada na Last FM e a duração da música, no formato JSON. O método é executado preenchendo a URL: http://ws.audioscrobbler.com/2.0/?method=track.getInfo&api_key=API_KEY&mbid=MBID&format=json. Como parâmetros do método temos API_KEY, que é a chave da sua aplicação e MBID que é o ID da música na aplicação.

5.1.3. Metadados

Primeiramente obteve-se os nomes de usuários que seriam utilizados no trabalho a partir do método user.getFriends. A partir de um usuário, obtemos o nome de seus amigos, e então buscamos o nome dos amigos dos amigos dos usuários, até que a base obtenha um número de usuários adequado ao estudo.

Como resultado obtemos a tabela USERS que possui o nome de 453569 usuários distintos. O metadados da tabela USERS se encontra na tabela 2.

Tabela 2 - Metadados USERS

TABELA USERS	
COLUNA	DESCRIÇÃO
USERNAME	Nome do usuário
FRIENDNAME	Nome do amigo do usuário

Dos usuários presentes na tabela USERS (tabela 2), foram selecionados aleatoriamente 28028 para o estudo. O método user.getInfo foi utilizado para os usuários selecionados, resultando na tabela USER_INFO, descrita na tabela 3.

Tabela 3 - Metadados USER_INFO

TABELA USER_INFO	
COLUNA	DESCRIÇÃO
USERNAME	Nome do usuário
REALNAME	Nome real do usuário
COUNTRY	País do usuário
GENDER	Genero do usuário
PLAYCOUNT	Quantidade de músicas escutadas pelo usuário
AGE	Idade do usuário

Com os usuários do estudo definidos, foi utilizado o método user.getTopTracks, para gerar a tabela USER_TOP_TRACK, descrita na tabela 4.

A classe *topTracks* [APENDICE B] foi utilizada para se obter as músicas mais escutadas pelos usuários selecionados.

Tabela 4- Metadados USER_TOP_TRACK

TABELA USER_TOP_TRACK	
COLUNA	DESCRIÇÃO
USERNAME	Nome do usuário
TRACK_NAME	Nome da música
PLAYCOUNT	Quantidade de vezes que a música foi escutada pelo usuário
MBID	ID da música na aplicação

O método user.getTopTracks pode retornar até 50 musicas para cada usuário, fazendo com que a tabela final obtivesse um total de 1374485 linhas, uma média de 49 músicas por usuário.

Selecionando todas as músicas diferentes presentes na tabela USER_TOP_TRACK, foi gerada a tabela CATALOGO_MUSICA, descrita na tabela 5.

Para se obter informações referentes às músicas encontradas na tabela USER_TOP_TRACK, foi utilizado o método track.getInfo, para evitar duplicidade a classe verifica se o MBID já está presente na tabela CATALOGO_MUSICA antes de buscar MBID no API.

Tabela 5 - Metadados CATALOGO_MUSICA

TABELA CATALOGO_MUSICA	
COLUNA	DESCRIÇÃO
MBID	ID da música na aplicação
TRACK_NAME	Nome da música
PLAYCOUNT	Quantidade de vezes que a música foi escutada na aplicação
LISTENERS	Quantidade de usuários diferentes que escutaram a música
ARTIST	Nome do artista da música
ARTISTID	ID do artista na aplicação
ALBUM	Álbum da música
ALBUMID	ID do álbum na aplicação
TAG1	1º TAG mais associado à música
TAG2	2º TAG mais associado à música
TAG3	3º TAG mais associado à música
TAG4	4º TAG mais associado à música
TAG5	5º TAG mais associado à música

O diagrama de banco de dados demonstrado na figura 4, ilustra as tabelas finais utilizadas no trabalho.

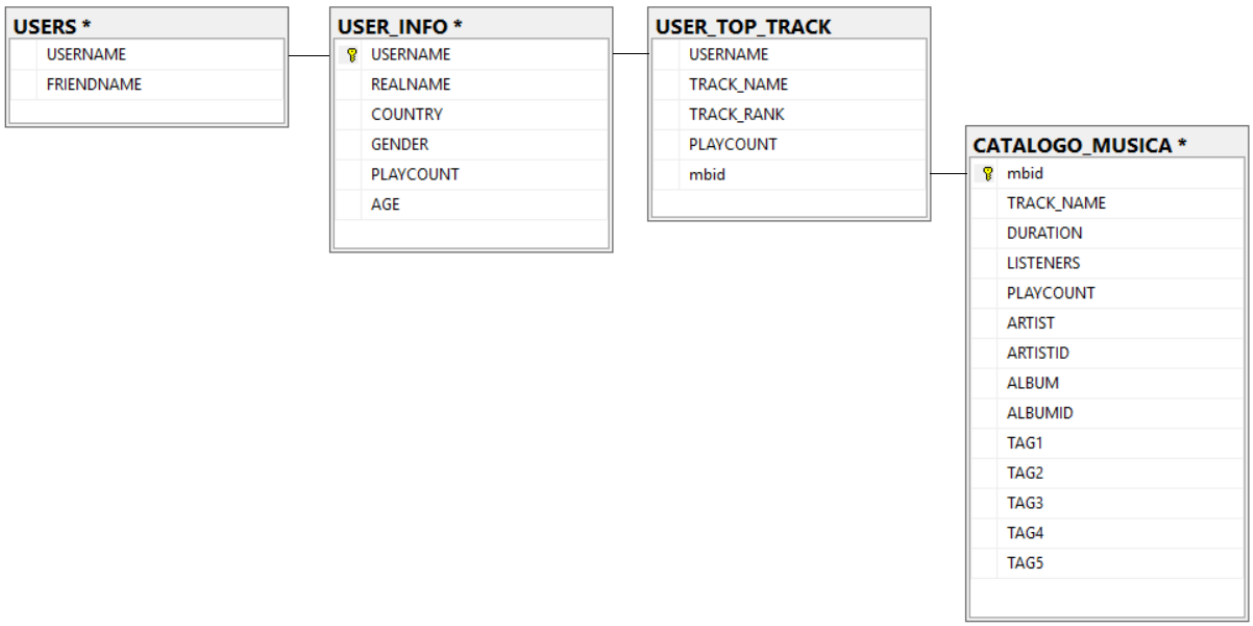

Figura 4- Diagrama de banco de dados

5.1.4. Preparação dos Dados

No processo de obtenção de dados e na análise dos dados, foram necessários alguns ajustes nas tabelas para que as recomendações de músicas fossem assertivas.

Como algumas pontuações geravam erros no código, todas as Strings retornadas pelos JSONs obtidos pelo API receberam um tratamento, alterando o caractere não aceito por um espaço vazio. Esse tratamento é facilmente implementado pela função *replaceAll* da biblioteca do JAVA.

Após a criação da tabela CATALOGO_MUSICA, notou-se que algumas músicas não foram encontradas na Last FM, pois saíram do catalogo da aplicação. Essas músicas foram removidas da tabela USER_TOP_TRACK por não ser possível obter as informações sobre elas. Removendo estas músicas, a tabela USER_TOP_TRACK ficou com 1084044 linhas, média de 38,6 músicas por usuário. Já a tabela CATALOGO_MUSICA teve um total de 255182 músicas distintas.

Mesmo verificando se o MBID já existe na tabela CATALOGO_MUSICA, antes de buscar o MBID no API da Last FM, alguns MBID apareceram duplicados na base. Verificando alguns casos foi possível notar, que para alguns MBID consultados no API, o MBID retornado no JSON era diferente, porém os outros dados se mantinham os mesmos tanto para o MBID consultado quanto na consulta do MBID retornado no JSON. Essa inconsistência entre o MBID pesquisado e

o retornado no JSON deve ter sido causada devido alguma atualização do catálogo da Last FM. Todos os MBID duplicados na base foram removidos por consultas manuais.

Para avaliar os sistemas de recomendação foram removidas 10 músicas de 1000 usuários distintos, que possuíam ao menos 49 músicas catalogadas na tabela USER_TOP_TRACK, as 10 músicas removidas tinham posições fixas no TRACK_RANK do usuário sendo que essas posições foram escolhidas aleatoriamente. As músicas removidas foram salvas em uma tabela chamada USER_TESTE, que possui os mesmos campos que a tabela USER_TOP_TRACK.

5.2. RECOMENDAÇÃO DE MÚSICAS

A partir dos dados gerados, foi possível criar sistemas de recomendações, com o objetivo de recomendar as músicas que foram removidas da tabela USER_TOP_TRACK.

5.2.1. Filtragem Colaborativa

A filtragem colaborativa aplicada no estudo utilizou o algoritmo KNN, a partir dos vizinhos mais próximos do usuário. Foram selecionadas as músicas mais comuns entre eles e estas foram recomendadas [APÊNDICE C].

Para se definir os vizinhos mais próximos do usuário, primeiramente selecionamos os 50 usuários que possuem mais músicas iguais na tabela USER_TOP_TRACK, a partir da query:

```
SELECT TOP 50 USERNAME
FROM USER_TOP_TRACK
WHERE MBID IN (SELECT MBID FROM USER_TOP_TRACK WHERE USERNAME = “ALVO”) GROUP BY USERNAME
ORDER BY COUNT(USERNAME) DESC
```

Onde “ALVO” é o nome do usuário que se deseja fazer recomendações.

Esta query ajuda a reduzir o tempo de processamento do sistema, uma vez que após ela definimos os n vizinhos mais próximos a partir da semelhança de cosseno, que quando aplicada a toda a base aumenta muito o tempo de processamento.

Como em nossa base temos apenas informação se a música está presente ou não na tabela USER_TOP_TRACK para o usuário (binário), a semelhança de cosseno adaptada para o estudo é obtida pela fórmula:

$$COS\Theta = \frac{Q_{ab}}{\sqrt{Q_a} x \sqrt{Q_b}}$$

Equação 9 - Semelhança do cosseno Adaptada

onde Q_{ab} é a quantidade de música que é encontrada na tabela USER_TOP_TRACK por ambos usuários, Q_a é a quantidade de música que é encontrada na tabela USER_TOP_TRACK para o usuário a, e Q_b é a quantidade de música que é encontrada na tabela USER_TOP_TRACK para o usuário b.

Quanto maior o valor do cosseno, mais relacionado estão os usuários.

Com os valores de cosseno obtidos em uma matriz relacionando usuário e cosseno, é feita uma ordenação utilizando o algoritmo buble sort [14] para se obter os K usuários mais próximos

Com os K usuários mais próximos definidos, é recomendado ao usuário um número M de músicas.

São selecionadas as m músicas que mais aparecem na tabela USER_TOP_TRACK para os K usuários vizinhos.

Estas m músicas são obtidas através da query:

```
SELECT TOP M MBID, TRACK_NAME
FROM (SELECT MBID, COUNT(MBID) AS CONTAGEM FROM (SELECT MBID FROM
USER_TOP_TRACK WHERE USERNAME IN("ALVOS") AND MBID NOT IN (SELECT
MBID FROM USER_TOP_TRACK WHERE USERNAME LIKE "ALVO")) AS T0
GROUP BY MBID
ORDER BY COUNT(MBID) DESC)
```

onde “ALVO” é o nome do usuário que se deseja recomendar músicas e “ALVOS” são os nomes dos usuários mais próximos do alvo. A query remove todas as músicas encontradas na tabela USER_TOP_TRACK para o alvo, evitando a recomendação de músicas que o usuário já tenha ouvido.

As músicas recomendadas são comparadas com as removidas anteriormente do usuário, para verificar se o sistema está acertando suas recomendações. Os resultados obtidos são exportados para uma planilha no formato CSV através do método exportToCSV presente na classe colaborative [APENDICE C].

5.2.2. Filtragem Baseada em Conteúdo

Para recomendar músicas baseadas no conteúdo, utilizamos as TAGs presentes no catálogo de música, o *playcount* da música e o artista da música.

Primeiramente, selecionamos todos os TAGs presentes em todas as músicas que encontramos, para o usuário que se deseja realizar a recomendação na tabela USER_TOP_TRACK.

Em seguida, filtramos na tabela CATALOGO_MUSICA apenas as músicas que possuem os 5 TAGs encontrados na primeira seleção, removendo as músicas que já foram escutadas pelo usuário alvo. As músicas selecionadas têm seu MBID guardado em uma matriz, junto com seu peso de recomendação inicial com o valor 1.

Para se determinar o peso de um artista, selecionamos a quantidade de músicas que ele é responsável e aparece na tabela USER_TOP_TRACK para o usuário alvo. Artistas que não aparecem em nenhuma das músicas selecionadas recebem o peso 1.

O peso final para cada música é encontrado multiplicando-se seu peso inicial pelo peso de artista e pelo seu play count encontrado na tabela CATALOGO_MUSICA.

Depois de se obter o peso final, as músicas são ordenadas de forma decrescente pelo seu peso através do algoritmo *quick sort*[14], presente na classe contentBased [APENDICE D]. As m músicas com maior peso são então recomendadas ao usuário.

De maneira semelhante à filtragem colaborativa, é realizada a comparação entre as músicas recomendadas e as músicas removidas anteriormente, e os resultados são exportados para uma planilha CSV.

5.2.3. Sistemas Hibridos

Combinando as técnicas de filtragem colaborativa e filtragem baseada em conteúdo, citadas anteriormente, foram criados dois sistemas de recomendação híbridos, um por peso e outro por troca.

Peso

O sistema de recomendação hibrido por peso, utilizou a mesma métrica de semelhança utilizada na filtragem colaborativa para a escolha dos N vizinhos mais próximos. Após a definição dos N vizinhos mais próximos, foram selecionadas todas as músicas da tabela USER_TOP_TRACK para os vizinhos selecionados e a contagem de vezes que a música apareceu nessa seleção se tornou o peso inicial da música nesse sistema.

Considerando apenas as músicas que receberam pesos iniciais, é selecionado o artista da música e a quantidade de vezes que ela foi escutada pela tabela CATALOGO_MUSICA. O sistema realiza a contagem de vezes que os artistas aparecem na tabela USER_TOP_TRACK para o usuário alvo e considera como peso do artista. Caso o artista não apareça, este recebe peso 1.

Por fim o sistema multiplica o peso inicial com o peso do artista e com a quantidade de vezes que a música foi escutada, gerando um peso final para as músicas. As músicas são ordenadas pelo seu peso final de maneira decrescente e as N músicas com maior peso são recomendadas ao usuário.

Troca

O sistema de recomendação híbrido por troca, define a técnica que irá utilizar, filtragem colaborativa ou filtragem baseada em conteúdo, a partir da média dos valores obtidos aplicando a fórmula de semelhança de cosseno para os N usuários com maiores nota, ou seja, se para um usuário alvo, seus vizinhos mais próximos tiverem uma nota média de semelhança acima de um valor pré-estabelecido, o sistema utilizará a filtragem colaborativa. No entanto, se a média for abaixo do valor pré-estabelecido o sistema utilizará a filtragem baseada em conteúdo. O valor de troca foi definido por diversas simulações desse valor, sendo que a simulação com os melhores resultados definiu seu valor final.

Após definir a técnica que será utilizada, filtragem colaborativa ou filtragem baseada em conteúdo, o sistema utiliza os mesmos parâmetros das técnicas descritas nos sistemas puros para realizar a recomendação de música.

Uma vez que as técnicas de sistemas de recomendação usadas neste trabalho foram descritas, na próxima seção serão apresentados resultados e análises comparativas relativos aos métodos implementados.

6. RESULTADOS

Utilizando os sistemas de recomendações criados, foram recomendadas 20 músicas para todos os usuários separados na tabela USER_TESTE. Como a tabela USER_TESTE tem 10 músicas para cada um dos 1000 usuários selecionados, recomendar 20 músicas significa um erro de, pelo menos, 50 % das recomendações, mesmo assim o valor de 20 músicas recomendadas foi escolhido, para se aumentar o número de verdadeiro-positivo dos resultados e obter uma melhor comparação entre os sistemas.

Para avaliar os sistemas de recomendações, foram comparadas as músicas recomendadas com as músicas separadas na tabela USER_TESTE, sendo que quanto mais músicas recomendadas estiverem na tabela USER_TESTE, melhor o sistema.

Uma musica recomendada ao usuário e que foi movida para a tabela de validação, é considerada um verdadeiro-positivo(VP); uma música recomendada ao usuário e que não está presente na tabela de validação é considerado um falso-positivo(FP); uma música que não foi recomendada e está na tabela de validação, é considerada um falso-negativo(FN), e uma musica que não foi recomendada e não está na tabela de validação, é considerada um verdadeiro-negativo(VN).

Foram simulados diversos valores de K com o objetivo de se determinar o melhor valor, para o algoritmo KNN utilizado na filtragem colaborativa. A tabela 6 mostra os resultados obtidos.

Tabela 6 - Resultados da filtragem colaborativa para diferentes valores de K.

K	Verdadeiro-Positivo	Falso-Positivo	Precisão	Recall
5	2412	17588	0,1206	0,2412
8	2580	17420	0,1290	0,2580
10	2637	17363	0,1318	0,2637
12	2646	17354	0,1323	0,2646
20	2680	17320	0,1340	0,2680
22	2776	17224	0,1388	0,2776
25	2745	17295	0,1353	0,2705

Observando os resultados, notamos que o melhor valor de K foi 22, e que ao aumentar o valor de K o sistema passa a errar mais, uma vez que ele passa a perder a sensibilidade das classificações para o usuário devido ao grande número de vizinhos considerados, conforme descrito na página 13.

A baixa precisão dos resultados foi causada pela grande quantidade de falso-positivo, esperado devido ao número de recomendações, e pelo grande catálogo do sistema, uma vez que é muito difícil retornar um item exato em um catálogo de 255182 itens.

A tabela 7 mostra os resultados obtidos para o sistema de recomendação onde foi utilizada filtragem baseada em conteúdo.

Tabela 7 - Resultados obtidos na filtragem baseada em conteúdo.

Método	Verdadeiro-Positivo	Falso-Positivo	Precisão	Recall
Content based	1979	18021	0,0989	0,1979

Comparando as tabelas 6 e 7, observamos que a filtragem colaborativa apresentou melhores resultados conforme indicado em outros estudos [12]. A baixa precisão do sistema se deve ao tamanho do catálogo e o número de músicas recomendadas.

Em relação à performance de tempo, o método de filtragem colaborativa também foi superior à filtragem baseada em conteúdo, uma vez que o sistema de filtragem colaborativa gerava recomendações a uma média de 5 segundos por usuário enquanto o sistema de filtragem baseada em conteúdo demorava em média 8 segundos.

A tabela 8 mostra os resultados obtidos para os sistemas de recomendação híbridos.

Para o sistema de troca, foi utilizado K=22, por ser o melhor valor obtido nas simulações anteriores. Como parâmetro de troca, foi utilizado um valor de cosseno médio de 0,002 por vizinho próximo. Esse valor representa um vizinho com aproximadamente 2 músicas iguais na tabela USER_TOP_TRACK. O valor foi escolhido porque os melhores resultados foram obtidos utilizando apenas o método de filtragem colaborativa, e com o valor de cosseno médio inferior a esse não seriam geradas recomendações por filtragem baseada em conteúdo aos usuários da tabela USER_TESTE, uma vez que todos estes tinham vizinhos com, ao menos, 1 música em comum. Desta forma os melhores resultados do sistema hibrido por troca, considerando que o sistema utilizou os dois tipos de recomendações, foi obtido com um cosseno médio de 0,002 por vizinho próximo.

Tabela 8 - Resultados obtidos nos sistemas de recomendação híbridos.

Método	Verdadeiro-Positivo	Falso-Positivo	Precisão	Recall
Peso	2108	17892	0,1054	0,2108
Troca	2674	17326	0,1337	0,2674

Comparando os sistemas de recomendação híbridos, o que utiliza o método de troca foi o que apresentou melhor resultado.

O sistema por peso acabou carregando a baixa precisão da filtragem baseada em conteúdo aos resultados. Deveriam ter sido feitas mais simulações dando um peso maior à parte de filtragem colaborativa para melhorar os resultados.

Era esperado por esse trabalho que o método de troca apresentasse o melhor resultado entre os sistemas, uma vez que quanto menor a semelhança entre os usuários, piores são as recomendações, e o sistema de troca tenta eliminar esse defeito passando a recomendar por filtragem baseada em conteúdo para os casos de usuários com pouca semelhança em relação aos vizinhos mais próximos. O sistema de troca poderia apresentar melhores resultados, quando comparado ao sistema de filtragem colaborativa puro, se na base de teste tivessem usuários com menores semelhança aos vizinhos próximos, desta forma seria possível realizar testes com menores valores de similaridade considerado para realizar a troca.

7. CONCLUSÃO

Devido ao grande aumento no volume de dados na internet os sistemas de recomendação estão, cada vez mais, sendo importantes no auxílio ao usuário na hora de encontrar o item desejado. Com as grandes bibliotecas musicais que os usuários têm a sua disposição hoje em dia, o estudo de recomendações nessa área se tornou bastante relevante.

A partir dos resultados obtidos no estudo, podemos concluir que para a filtragem colaborativa, o K=22 foi aquele que demonstrou melhor resultado, e que o sistema de recomendação que utiliza filtragem colaborativa possui resultados melhores em comparação com os sistemas de recomendação baseada em conteúdo. Para melhorar o desempenho do sistema de recomendação que utilizou filtragem baseada em conteúdo, podemos utilizar outros campos referente às músicas escutadas pelos usuários, como álbum por exemplo.

Esperava-se melhores resultados dos sistemas de recomendação híbridos, quando comparados com os demais métodos, uma vez que a união de dois métodos procura eliminar as desvantagens de cada método separado. A melhora esperada, porém, não foi observada. O método que utiliza apenas a filtragem colaborativa, foi o que apresentou melhor resultado, indicando que os fatores que definem a combinação dos métodos não tenham sido bem definidos.

Os valores encontrados na avaliação dos sistemas de recomendação criados para o estudo são baixos, porém é difícil inferir que os itens classificados como falso-positivo não são de interesse do usuário, uma vez que não possuímos a avaliação do usuário para todos os itens catalogados, sabemos apenas as 50 músicas mais escutadas dentro de um catálogo que possui 255182 músicas distintas.

8. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Ricci, F, Rokach, L, Shapira, B. *Recommender Systems Survey*. Springer, 2010.

[2] Bobadilha, J., Ortega, F., Hernando, A., Gutiérrez, A. *Recommender systems survey*. Elsevier, 2013.

[3] Almazro, D., et al. *A survey paper on recommender systems*. 2010.

[4] Vekariya, V., Kulkarni, G. *Hybrid recommender systems: Survey and experiments*. *Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, Bangkok, 2012, pp. 469-473.

[5] Isinkaye, F., Folajimi, Y., Ojokoh, B. *Recommendation systems: Principles, methods and evaluation*. Elsevier, 2015.

[6] Shani, G., Gunawardana, A. *Evaluating Recommender Systems.* Microsoft research, 2009.

[7] Introdução ao JSON. Disponível em: <https://www.json.org/json-pt.html> Acesso em: 09 de março de 2019.

[8] JAVA SE Technologies - Database. Disponível em: <https://www.oracle.com/technetwork/java/javase/jdbc/index.html> Acesso em: 25 de fevereiro de 2019.

[9] Mecanismo de Banco de Dados do SQL Server. Disponível em: <https://docs.microsoft.com/pt-br/sql/database-engine/sql-server-database-engine-overview?view=sql-server-2017> Acesso em: 09 de março de 2019.

[10] SQL Server Management Studio. Disponível em: <https://docs.microsoft.com/pt-br/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2014> Acesso em: 09 de março de 2019.

[11] Sobre a Last.FM. Disponível em: <https://www.last.fm/pt/about> Acesso em: 25 de fevereiro de 2019.

[12] Kaitila, Juuso Kaitila. A contente-based music recommender system. 65 f. Dissertação (Mestrado em Ciências da computação) Faculty of Natural Sciences, University of Tampere, Tampere, 2017.

[13] Baeza-Yates, R. *Recuperação de informação: Conceitos e tecnologias das máquinas de busca*. 2ª ed. Bookman, 2013.

[14] Cormen, T, Leiserson, C, Rivest, R. *Algoritmos: Teoria e Prática*. 2ª ed. Campus, 2002.

9. APÊNDICE A – Classe SQL, conexão e query

```
public class SQL {

    private String host;
    private String user;
    private String pass;
    private String database;

    public Connection c;

    public SQL ( String host, String database, String user, String pass ) {
        this.pass = pass;
        this.user = user;
        this.host = host;
        this.database = database;
    }

    /**
     * Método que estabelece a conexão com o banco de dados
     *
     * @return True se conseguir conectar, falso em caso contrário.
     */
    public boolean connect() {
        boolean isConnected = false;

        String url;
        String portNumber = "1433";
        String userName   = this.user;
        String passName   = this.pass;
        url = "jdbc:sqlserver://"+ this.host+":" +portNumber + ";databaseName="
+this.database;

        try {

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
            this.c = DriverManager.getConnection(url,userName, passName);
            isConnected = true;
        } catch( SQLException e ) {
            e.printStackTrace();
            System.out.println(e.getMessage());
            isConnected = false;
        } catch ( ClassNotFoundException e ) {
            e.printStackTrace();
            System.out.println(e.getMessage());
            isConnected = false;
        } catch ( InstantiationException e ) {
            e.printStackTrace();
            System.out.println(e.getMessage());
            isConnected = false;
        } catch ( IllegalAccessException e ) {
            e.printStackTrace();
            System.out.println(e.getMessage());
            isConnected = false;
        }

        return isConnected;
    }
```

```
    /**
     * Método que estabelece a desconexão com o banco de dados
     * @return True se conseguir desconectar, falso em caso contrário.
     */
    public boolean disconnect() {
        boolean isConnected = false;

        String url;
        String portNumber = "1433";
        String userName   = this.user;
        String passName   = this.pass;
        url = "jdbc:sqlserver://"+ this.host+":" +portNumber + ";databaseName="
+this.database;

        try {
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
            this.c = DriverManager.getConnection(url,userName, passName);
            this.c.close();
            isConnected = true;
        } catch( SQLException e ) {
            System.out.println(e.getMessage());
            isConnected = false;
        } catch ( ClassNotFoundException e ) {
            System.out.println(e.getMessage());
            isConnected = false;
        } catch ( InstantiationException e ) {
            System.out.println(e.getMessage());
            isConnected = false;
        } catch ( IllegalAccessException e ) {
            System.out.println(e.getMessage());
            isConnected = false;
        }
        return isConnected;
    }
//.....................................................................................................................
/*
     * Método que insere valores obtidos do parser na tabela USER_TOP_TRACK
*/
public void insertTopTrack( String USERNAME, String TRACKNAME,String TRACKRANK,String
PLAYCOUNT, String mbid ) {
        Statement st;
        int rs;

        try {
            st = this.c.createStatement();
            String insert= "INSERT INTO USER_TOP_TRACK"
                   + "(USERNAME, TRACK_NAME, TRACK_RANK,PLAYCOUNT, mbid) VALUES ('"
+USERNAME+"','"+TRACKNAME+"','"+TRACKRANK+"','"+PLAYCOUNT+"','"+ mbid+"')";
             System.out.println(insert);
            rs = st.executeUpdate(insert);


        } catch ( SQLException e ) {
            e.printStackTrace();
        }

        return;
    }
//.....................................................................................................................
```

10. APÊNDICE B – Classe topTracks

```
    //Classe responsável pela preenchimento da tabela USER_TOP_TRACK
public class topTracks {
      static ArrayList<String> nomess = new ArrayList<String>();

public static void main(String[] args) {

    HttpURLConnection connection = null;
      SQL SQL = new SQL("SERVER", "DATABASE","USER", "PASSWORD");
      SQL.connect();
//Consulta todos os usuários na tabela USERS que ainda não estão na tabela
USER_TOP_TRACK
    nomess =SQL.usuariosTopTrack();
      int j=0;
//Realiza a interação para obter informações de todos usuários retornados
      for (int i=0;i<nomess.size();i++ )
      {
      try
      {
    // Realiza a conexão
    URL url = new
URL("http://ws.audioscrobbler.com/2.0/?method=user.gettoptracks&user="+nomess.get(j)+
"&api_key=APIKEY&format=json");
          connection = (HttpURLConnection) url.openConnection();

          connection.setUseCaches(false);
          connection.setDoInput(true);

          // Verifica a resposta HTTP, 200=OK
          if( connection.getResponseCode() != 200 )
          {
             System.out.println("não deu 200\n deu:"+ connection.getResponseCode());
          }
      // Insere JSON obtido em um StingBuilder
          BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
          StringBuilder contents = new StringBuilder();

          String line;
          while( (line = in.readLine()) != null)
          {
              contents.append(line);
          }
          in.close();
          // Parse do documento JSON.
             String insere = parserString (contents.toString(),nomess.get(j),SQL);
             j ++;
      }
      catch( Throwable t)
      {}
      finally
      {
          if( connection != null ) connection.disconnect();
      }

}

            SQL.disconnect();
}
```

```
// Realiza o parser do documento JSON obtendo apenas os resultados que interessa.
public static String parserString(String jsonString, String USERNAME,SQL SQL) throws
ParseException, org.json.simple.parser.ParseException
{
	JSONObject json;
	try {
		json = new JSONObject(jsonString);
		JSONObject responseData = json.getJSONObject("toptracks");
	 JSONArray jArray = responseData.getJSONArray("track");
	 int n = jArray.length();

	 for (int i = 0; i < n; ++i) {
	  JSONObject person = jArray.getJSONObject(i);
	 try {
		String MUSICA = person.getString("name");
		MUSICA = MUSICA.replaceAll ( "'","");
//Insere dados na tabel USER_TOP_TRACK
		 SQL.insertTopTrack(USERNAME,
MUSICA,Integer.toString(i),person.getString("playcount"),person.getString("mbid"));
	    } catch (Exception e) {
		System.err.println(e);
	    }
		 }
	} catch (JSONException e) {
		e.printStackTrace();
	}

    return "ok";
}
}
```

11. APÊNDICE C – Classe colaborative

```
public class colaborative {

public static void main(String[] args) {

//Cria um objeto SQL, responsável por chamar os métodos da classe SQL
      SQL SQL = new SQL("SERVER", "BANCO DE DADOS","USER", "PASSWORD");
      SQL.connect();
//Quantidade de músicas recomendadas para cada usuário
      int numMusicas = 20;
//Cria lista de usuários que se deseja recomendar músicas
    ArrayList<String> alvo = SQL.USUARIOS_TEST();
//Quantidade de vizinhos próximos
      int knn = 12;
//Cria matriz de String para exportar CSV com as recomedações
      String Recomendacoes [][] = new String[(alvo.size())*numMusicas][16];
      String temp [][] = new String[numMusicas][16];
      String csv ="DIRETÓRIO ARQUIVO CSV”;
//Realiza iteração para recomendar músicas a todos usuários da lista alvo,
preenchendo matriz consolidada de recomendações chamada recomendacoes
      for (int i = 0; i < alvo.size(); i++) {
//Realiza filtragem colaborativa para recomendar músicas
            temp = colabor(alvo.get(i),SQL,knn,numMusicas);
            int k = i*numMusicas;
            for(int m = 0; m < numMusicas; m++) {
                  for(int j = 0; j < 16; j++) {
                  Recomendacoes[k+m][j]= temp[m][j];
                  }
            }
      }
//Chama método que gera arquivo .CSV
      boolean a = exportToCSV(Recomendacoes,csv,(alvo.size())*numMusicas);
      SQL.disconnect();

}
public static String[][] colabor(String alvo, SQL SQL, int knn, int numMusicas) {
    String alvos ="";
    ArrayList<String> top = new ArrayList<String>();
//Retorna os 50 usuários com mais músicas iguais ao usuário alvo na tabela
USER_TOP_TRACK
      top = SQL.topMatch(alvo);
//Retorna numero de músicas do usuário alvo na tabela USER_TOP_TRACK
      int numTrack =SQL.numTrack(alvo);
      int numTrack2 = 0;
      int Similar = 0;
      double topK [][]= new double [top.size()][2];
      double aux =0;
//Laço que calcula a semelhança do cosseno para os usuários da lista top
      for (int i = 0; i < top.size(); i++) {
            numTrack2 =SQL.numTrack(top.get(i));
            Similar = SQL.numSimilar(alvo ,top.get(i));
            topK[i][0]=i;
            topK[i][1]=cosineSimilarity(Similar,numTrack,numTrack2);
          }

//buble sort, ordena lista top pela semelhança do cosseno obtida
    boolean troca = true;
    while (troca) {
        troca = false;
```

```
    for(int i = 0; i<top.size()-1; i++){
            if(topK[i][1] < topK[i+1][1]){
                aux = topK[i][1];
                topK[i][1] = topK[i+1][1];
                topK[i+1][1]= aux;
                aux = topK[i][0];
                topK[i][0] = topK[i+1][0];
                topK[i+1][0]= aux;
                troca = true;
            }
      }
    }
//Se o numero de usuários da lista top for menor que o KNN definido
      if ((knn) > top.size()) {
//Laço que cria uma String com os usuários próximos formatada para consulta SQL
             for (int i = 1; i < top.size()-1; i++)
                   alvos += "'"+top.get((int) topK[i][0])+"' ,";
                   alvos += "'"+top.get((int) topK[top.size()-1][0])+"'";
                      System.out.println(alvos);
//Consulta SQL que retorna as músicas que mais aparecem na tabela USER_TOP_TRACK,
para os vizinhos próximos definidos, no caso de empate a música com maior PLAYCOUNT é
selecionada. A quantidade de músicas retornadas é definida pela variável numMusicas
                   return SQL.recomendacoes(alvos,alvo,numMusicas);
      }
      else {
//Laço que cria uma String com os usuários próximos formatada para consulta SQL
             for (int i = 1; i < knn; i++)
                   alvos += "'"+top.get((int) topK[i][0])+"' ,";
                   alvos += "'"+top.get((int) topK[knn+1][0])+"'";
                      System.out.println(alvos);
                   return SQL.recomendacoes(alvos,alvo,numMusicas);
      }
}
//Método que gera arquivo .CSV
public static boolean exportToCSV(String[][] tableToExport,
        String pathToExportTo,int numeroLinhas) {

    try {
        FileWriter csv = new FileWriter(new File(pathToExportTo));
//Escreve na primeira linha o cabeçalho da tabela
csv.write("mbid,TRACK_NAME,DURATION,LISTENERS,PLAYCOUNT,ARTIST,ARTISTID,ALBUM,ALBUMID
,TAG1,TAG2,TAG3,TAG4,TAG5,USER");
        csv.write("\n");

        for (int i = 0; i < numeroLinhas; i++) {
            for (int j = 0; j < 16; j++) {
                csv.write(tableToExport[i][j]+ ",");
            }
            csv.write("\n");
        }

        csv.close();
        return true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}

//Realiza o cálculo da semelhança do coseno
public static double cosineSimilarity(int dotProduct, int a, int b)
```

```
{

        double cosineSimilarity =0;
        cosineSimilarity = (dotProduct / (Math.sqrt(a) * Math.sqrt(b)));
        return cosineSimilarity;
}
}
```

12. APÊNDICE D – Classe content

```
public class contentBased {

public static void main(String[] args) {

//Cria um objeto SQL, responsável por chamar os métodos da classe SQL
      SQL SQL = new SQL("SERVER", "BANCO DE DADOS","USER", "PASSWORD");
      SQL.connect();
//Quantidade de músicas recomendadas para cada usuário
      int numMusicas = 20;
//Cria lista de usuários que se deseja recomendar músicas
    ArrayList<String> alvo = SQL.USUARIOS_TEST();
//Cria matriz de String para exportar CSV com as recomendações
      String Recomendacoes [][] = new String[(alvo.size())*numMusicas][16];
      String temp [][] = new String[numMusicas][16];
      String csv ="Diretório do arquivo .CSV " ;
//Realiza iteração para recomendar músicas a todos usuários da lista alvo,
preenchendo matriz consolidada de recomendações chamada recomendacoes
      for (int i = 0; i < alvo.size(); i++) {
            temp = content(alvo.get(i),SQL,numMusicas);
            int k = i*numMusicas;
            for(int m = 0; m < numMusicas; m++) {
                  for(int j = 0; j < 16; j++) {
                  Recomendacoes[k+m][j]= temp[m][j];
                  }
            }
      }

      boolean a = exportToCSV(Recomendacoes,csv,(alvo.size())*numMusicas);
      SQL.disconnect();

}
//Método que utiliza filtragem baseada em conteúdo para realizar recomendações,
recebe como argumento o usuário que se deseja realizar recomendação(alvo), o objeto
SQL(SQL) e a quantidade de músicas que se deseja recomendar (numMusicas).
public static String[][] content(String alvo, SQL SQL, int numMusicas) {
    String alvos ="";
//Um ArrayList é criado porque não sabemos o numero de musicas que a consulta
resultará.
    ArrayList<Object> top = new ArrayList<Object>();
//Cria uma lista com todas as músicas do catálogo em que as 5 TAGs estejam presentes
em alguma das músicas do usuário alvo na tabela USER_TOP_TRACK. O Objeto retorna o
mbid da música, o número 1 como peso inicial para a música, o ARTISTID e o PLAYCONT
da música.
      top = SQL.topTAG(alvo);
      String vetor [][]= new String [top.size()][4];
//Laço que transforma o ArrayList em uma matriz de String, para ganhar performance,
sendo que na posição 0 é colocado o mbid, na posição 1 é colocado o peso, na posição
2 é colocado o ARTISTID e na posição 3 é colocado o PLAYCOUNT
      for (int i = 0; i < top.size(); i++) {
            vetor [i][0]= (String) ((Object[])top.get(i))[0];
            vetor [i][1]= (String) ((Object[])top.get(i))[1];
            vetor [i][2]= (String) ((Object[])top.get(i))[2];
            vetor [i][3]= (String) ((Object[])top.get(i))[3];
          }
      ArrayList<String> artist = new ArrayList<String>();
//Realiza a contagem de vezes que um artista aparece para o usuário alvo na tabela
USER_TOP_TRACK. Retorna concatenado em uma String o ARTISTID e nos últimos dois
caracteres a quantidade de vezes que o artista aparece
      artist = SQL.artist(alvo);
```

```
      String artistas [][]= new String [artist.size()][2];
//Para ganhar performance, transforma o ArrayList em uma matriz de String, colocando
na posição 0 o ARTISTID e na posição 1 a quantidade de vezes que o artista aparece.
      for (int i = 0; i < artist.size(); i++) {
            artistas [i][0]= artist.get(i).substring(0, artist.get(i).length()-2);
            artistas [i][1] = artist.get(i).substring(artist.get(i).length()-2);
      }

//Laço que compara os ARTISTID da matriz artistas com a matriz vetor, quando o
ARTISTID é igual, a quantidade de vezes que o artista aparece é multiplicada pelo
peso inicial da música.
      for (int i = 0; i < top.size(); i++) {
            for (int j = 0; j < artist.size(); j++) {
            if (artistas[j][0].equals(vetor [i][2])) {
vetor[i][1]=String.valueOf(Integer.parseInt(vetor[i][1])*Integer.parseInt(artistas[j]
[1]));
            }
          }
//Peso final de cada musica é obtido multiplicando-se o peso resultante pelo
PLAYCOUNT de cada música
            vetor[i][1]=String.valueOf(Integer.parseInt(vetor[i][1]) *
Integer.parseInt( vetor[i][3]));
      }
//Método responsável pela ordenação da matriz pelo peso final de cada música
      quickSort(vetor,0,vetor.length-1);
      for (int i = 0; i < top.size(); i++) {
               System.out.println("contains" +vetor[i][1]);
      }
//Retorna as N musicas com maior peso
          for (int i = 0; i < numMusicas-1; i++)
                alvos += "'"+vetor[i][0]+"' ,";
                alvos += "'"+vetor[numMusicas-1][0]+"'";
                return SQL.recomendacoesContent(alvos,alvo,numMusicas);
}

//Algoritmo quick sort, adaptado para matriz[N][2]
private static void quickSort(String[][] vetor, int inicio, int fim) {
    if (inicio < fim) {
          int posicaoPivo = separar(vetor, inicio, fim);
          quickSort(vetor, inicio, posicaoPivo - 1);
          quickSort(vetor, posicaoPivo + 1, fim);
    }
}
private static int separar(String[][] vetor, int inicio, int fim) {
       String troca = "";
    int pivo = Integer.parseInt(vetor[inicio][1]);
    int i = inicio + 1, f = fim;
    while (i <= f) {
          if (Integer.parseInt(vetor[i][1]) >= pivo)
                i++;
          else if (pivo > Integer.parseInt(vetor[f][1]))
                f--;
          else {
                troca = String.valueOf(vetor[i][0]);
                vetor[i][0] = vetor[f][0];
                vetor[f][0]= troca;
                troca = String.valueOf(vetor[i][1]);
                vetor[i][1] = vetor[f][1];
                vetor[f][1]= troca;
                i++;
                f--;
```

```
            }
        }
        vetor[inicio][1] = vetor[f][1];
        vetor[f][1] = String.valueOf(pivo);
        return f;
    }
    }
```