Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Geração de Recomendações Usando Filtragem Colaborativa

Eduardo Delgado Coloma Bier

Monografia Final

MAC 0499 - Trabalho de Formatura Supervisionado

São Paulo
Dezembro de 2019

Resumo

Há cada vez mais técnicas de geração de recomendações sendo criadas. Tanto a academia quanto a indústria mostram grande interesse na área, investindo pesadamente no desenvolvimento de novas técnicas. O objetivo deste trabalho é estudar algumas dessas técnicas e formas de análise. Mais especificamente, este trabalho se interessa por implementar e analisar alguns algoritmos clássicos de filtragem colaborativa: filtragem colaborativa baseada em usuários, em items e passeios aleatórios. Foram utilizados dados do MovieLens para gerar recomendações de filmes para usuários e avaliar os algoritmos implementados: filtragrem colaborativa baseada em itens, baseada em usuários, passeio aleatório e passeio aleatório enviesado. Os resultados mostraram que os algoritmos implementados não tiveram bons desempenhos na base dados utilizada, mas revelaram informações interessantes sobre os efeitos das medidas de similaridade e número de recomendações sobre os resultados dos experimentos.

Palavras-chave: filtragem colaborativa, recomendações, grafos, neo4j.

Sumário

1 Introdução **1**

1.1 Motivação . . . 1

1.2 Objetivos . . . 2

1.3 Estrutura do Trabalho . . . 2

2 Conceitos **3**

2.1 Sistemas de Recomendação . . . 3

2.1.1 Usuário . . . 6

2.1.2 Item . . . 7

2.1.3 Transação . . . 7

2.1.4 Recomendação . . . 8

2.2 Filtragem Colaborativa . . . 9

2.2.1 Filtragem Colaborativa Baseada em Usuários . . . 9

2.2.2 Filtragem Colaborativa Baseada em Itens . . . 10

2.2.3 Passeio Aleatório . . . 11

2.3 Medidas de similaridade . . . 12

2.3.1 Similaridade de Cossenos . . . 12

2.3.2 Coeficiente de Pearson . . . 13

2.4 Avaliando Sistemas de Recomendação . . . 13

2.4.1 Métricas . . . 14

3 Implementação **17**

3.1 Dados Usados . . . 17

3.2 Neo4j . . . 18

3.3 Representação dos Dados . . . 18

3.3.1 Implementação dos Algoritmos . . . 20

3.4 Descrição dos Experimentos . . . 22

4 Resultados **25**

5 Conclusão **31**

Referências Bibliográficas **33**

Capítulo 1

Introdução

1.1 Motivação

A enorme quantidade de informação e conteúdo disponível em grandes plataformas digitais pode ser ofuscante para seus usuários. Em plataformas de streaming de filmes, por exemplo, ter uma grande quantidade de videos disponíveis para se assistir não se traduz diretamente em boas escolhas por parte do usuário [10]. Isso significa que embora um usuário tenha um grande leque de possíveis filmes para assistir, escolher um deles pode se tornar uma tarefa muito difícil e levá-lo a escolher um filme do qual não irá gostar.

Para enfrentar esse problema, foram criados Sistemas de Recomendação, cujo principal objetivo é gerar sugestões de itens para serem consumidos por um usuário a partir de seu perfil. Assim, com essa lista de recomendações em mãos, usuários são mais capazes de encontrar itens que realmente os interesse. Isso faz com que não só a experiência do usuário na plataforma melhore, mas também se traduz no aumento do consumo de itens, o que é extremamente valioso para as plataformas.

A capacidade de mostrar os produtos certos para as pessoas certas é essencial para sites de comércio eletrônico. Não se pode esperar que alguém seja capaz de encontrar os poucos itens que o interessem dentre um catálogo composto por milhões de itens. Assim, não é surpreeendente que empresas como a *Amazon*, que em 2016 tinha mais de 350 milhões de produtos disponíveis em sua plataforma [1], se preocupem tanto em aprimorar seu Sistema de Recomendação.

Além disso, Sistemas de Recomendação podem também fazer parte central das interfaces dos sites. A plataforma de streaming *Netflix*, por exemplo, cujo catálogo conta com milhares de filmes e séries, compõe sua página principal com uma sequência de listas geradas de acordo com o usuário logado. Assim, a navegação do usuário no site é completamente personalizável graças ao sistema recomendador implementado.

Assim, o desenvolvimento de Sistemas de Recomendação capazes de gerar boas recomendações é vital para muitas aplicações comerciais e grandes empresas estão preparadas para investir pesadamente nisso. Em 2009, por exemplo, a *Netflix* ofereceu um prêmio de U$ 1

milhão para quem conseguisse melhorar a qualidade de suas recomendações em 10% [7].

Para que um bom Sistema de Recomendação seja desenvolvido, é necessário não só decidir quais algoritmos serão utilizados, mas também quais informações serão exploradas por eles. Além disso, o domínio e o padrão de consumo dos usuários pode também afetar a performance do sistema. Trata-se, portanto, de uma tarefa extremamente desafiadora, uma vez que existem inúmeros métodos e formas de gerar recomendações.

Um dos principais métodos para gerar recomendações é a filtragem colaborativa, em que um item é ou não recomendado a um usuário com base no comportamento de outros usuários. Assim, a filtragem colaborativa leva em consideração o histórico de compra de outros usuários para determinar outros itens que possam interessar o usuário em questão.

Dependendo das informações exploradas, um algoritmo de filtragem colaborativa pode ser: baseado em items, baseado em usuários ou híbrido. Neste trabalho, para representar cada uma dessas categorias de algoritmos de filtragem colaborativa, foram implementados três algoritmos para gerar recomendações de filmes usando a base de dados do MovieLens. Esta base conta com 100 mil avaliações feitas por 943 usuários sobre 1682 filmes.

1.2 Objetivos

- Estudar algoritmos e estruturas de dados para geração de recomendações;
- Implementar algoritmos de recomendação usando filtragem colaborativa;
- Avaliar e comparar os algoritmos implementados.

1.3 Estrutura do Trabalho

O capítulo 2 apresenta a fundamentação teórica para o desenvolvimento de um sistema de recomendação. Aqui são definidos os principais conceitos e são descritos os algoritmos a serem implementados.

O capítulo 3, por sua vez, descreve as ferramentas utilizadas para a implementação do sistema, os dados utilizados e os experimentos realizados.

Os resultados dos experimentos são discutidos no capítulo 4.

Por fim, o capítulo 5 traz as considerações finais e aponta possíveis trabalhos futuros.

Capítulo 2

Conceitos

2.1 Sistemas de Recomendação

Sistemas de recomendação (SRs) são sistemas de softwares cuja principal função é gerar recomendações para seus usuários de forma personalizada. O teor dessas recomendações varia de acordo com o domínio do SR. Em uma plataforma de streaming de músicas, por exemplo, um SR pode sugerir músicas, artistas e playlists para um usuário escutar, enquanto em um site de comércio eletrônico, um usuário receberia sugestões de produtos para comprar. De forma geral, chama-se de item aquilo que um SR recomenda e de usuário a pessoa que está recebendo tais recomendações. Uma interação entre um usuário e um item, por sua vez, é chamada de transação.

As recomendações geradas são personalizadas, ou seja, as preferências de um usuário são levadas em conta na hora de gerá-las. Para fazer isso, é necessário que um SR seja capaz de traçar e armazenar o perfil de um usuário para identificar seus gostos. Isso pode ser feito de forma explícita, onde o programa pede ao usuário que indique suas preferências, ou de forma implícita, onde o comportamento do usuário é monitorado para determinar seu perfil.

Note que embora recomendações não personalizadas não sejam comumente exploradas por pesquisas em SRs, elas são frequentemente encontradas em sites para complementar recomendações personalizadas. A Amazon, por exemplo, além de implementar diferentes algoritmos para gerar recomendações personalizadas, apresenta também a lista de produtos mais vendidos que, apesar de ser não-personalizada, é relevante para um usuário médio.

Ao se falar em SRs, é importante destacar que existem dois principais agentes com objetivos distintos. De um lado estão os usuários, que usam o SR principalmente para encontrar itens que os interessem com maior facilidade. Do outro, estão as empresas que implementaram um SR em sua aplicação, cujo principal objetivo é fazer com que usuários consumam mais itens.

Com o intuito de facilitar o entendimento das necessidades que um SR pode suprir, será agora descrito um caso de uso de um SR. Imagine uma plataforma de streaming de filmes chamada *FilmesJá*, onde usuários pagam uma mensalidade para ter acesso ilimitado aos

milhares de filmes disponíveis no site. Devido a grande quantidade de videos disponíveis no catálogo e a ausência de uma ordenação deles, João, um dos usuários do *FilmesJá*, tem uma enorme dificuldade de encontrar um filme que lhe interesse. Por isso, João acaba usando a plataforma apenas quando sabe exatamente o que quer assistir e se questiona se de fato deveria estar pagando pelo serviço.

Essa situação não é sustentável a longo prazo, já que, assim como João, outras pessoas estão tendo os mesmos problemas e a *FilmesJá* corre o risco de perder sua base de usuários. Assim, é preciso que a plataforma aumente o nível de satisfação de seus clientes, fazendo com que eles continuem assinando o serviço. Isso significa que a usabilidade do sistema deve ser melhorada. Além disso, a quantidade e satisfação com os filmes que seus usuários assistem também deve aumentar. A fim de remediar a situação, os desenvolvedores do site decidem implementar um SR.

Um SR ajuda uma plataforma a implementar uma série de funcionalidades que podem beneficiar seus usuários. Herlocker et al. [3] identificaram algumas funcionalidades comumente relacionadas a SR, que serão agora discutidas. Quando aplicável, o exemplo do *FilmesJá* será usado.

Em primeiro lugar, está a funcionalidade central de qualquer SR: possibilitar usuários a encontrar itens bons, isto é, que lhes interessem. No contexto do *FilmesJá*, isso significa que ao invés de se deparar com uma lista aleatória de filmes para assistir, João passa a receber uma lista com alguns filmes que ele provavelmente irá gostar. Tal lista é normalmente ordenada em ordem decrescente de quanto o SR estimou que o usuário fosse gostar do item. Em alguns casos, essas estimativas são exibidas para o usuário.

Na maior parte das aplicações, incluindo a *FilmesJá*, deixar de encontrar alguns itens bons em troca de filtrar a maioria dos itens ruins é o bastante para satisfazer os usuários de uma plataforma. No entanto, em alguns domínios, um SR deve ser capaz de encontrar todos os itens relevantes. Isso é essencial para advogados procurando precedentes para um caso, por exemplo. Nessa situação, eles estão dispostos a gastar uma grande quantidade de tempo e energia para encontrar um caso perfeito, então usar um SR que possivelmente filtre um desses casos não é vantajoso.

SRs não se limitam a produzir apenas uma lista de itens para um usuário consumir. Uma série de listas podem ser produzidas, cada uma com um critério diferente. Um desses critérios pode ser montar listas com itens que combinem e façam sentido ser consumidos juntos. Por exemplo, o SR do *FilmesJá* pode montar uma lista com filmes da Pixar para João depois de perceber que ele gosta deste tipo de filme.

Outro tipo de lista que pode ser montada por um SR é uma recomendação de sequência de itens. Tal tipo de lista é muito comum em aplicações de streaming de música como o Spotify, que, a partir de uma música, são capazes de produzir um "rádio" que toca músicas semelhantes à música original. No caso de filmes, tais listas aparecem como sugestões do que assistir na sequência ao final dos créditos.

Além de montar listas, um SR pode também destacar itens dentro delas, adicionando

anotações em contexto. No exemplo do *FilmesJá*, isso seria equivalente a enfatizar um filme colocando na própria lista a estimativa de quanto um usuário gostaria dele. Outro exemplo de anotação seria incluir uma lista de amigos que também assistiram àquele filme.

Em alguns casos, usuários não tem o desejo imininente de consumir um item, mas estão na plataforma só olhando o que há disponível. Nesses casos, o SR é necessário não só para mostrar itens que interessem o usuário, mas também para possivelmente convencê-lo a consumir algo. Nessa situação, talvez ainda mais importante seja a qualidade da interface da plataforma, já que a experiência do usuário deve ser positiva para que "só olhar" seja algo prazeroso.

Uma outra funcionalidade essencial de um SR é a capacidade do usuário de expressar suas opiniões e alguns usuários valorizam muito isso. Esse desejo de se expressar pode ser puramente pelo ato em si, mas também pode ir além disso. Alguns usuários querem contribuir com informações para o bem da comunidade, enquanto outros podem fazê-lo com o intuito de influenciar a decisão de outras pessoas. Além disso, ao fornecer feedback para o SR, seu próprio perfil é enriquecido, aumentando a qualidade das recomendações que recebe.

Voltando para o exemplo do *FilmesJá*, João tem múltiplas razões para querer avaliar um filme. Ao dar nota para um filme, João sabe que as recomendações que receberá serão feitas com mais dados e, por isso, serão, em geral, de maior qualidade. Além disso, outras pessoas poderão ver o que ele achou do filme e levar isso em conta na hora de escolher algo para assistir.

Por fim, existe ainda mais uma razão para João querer avaliar um filme: testar o SR. Muitos usuários não confiam em recomendadores automaticamente e procuram fazer experimentos para verificar sua qualidade. Alguns sistemas incluem ferramentas para aumentar a confiança desses usuários, como incluir explicações junto das recomendações que um usuário recebe. Assim, sabendo da origem da recomendação, um usuário se sente mais disposto em confiar no SR. Um exemplo disso seria João receber uma lista composta por filmes da Pixar intitulada "Porque você assistiu Toy Story".

Do ponto de vista da plataforma, a implementação de um SR também pode ser extremamente valiosa. O fato de usuários receberem recomendações de produtos que lhe interessam se traduz diretamente no aumento do consumo de itens, já que eles são expostos a itens mais relevantes para seu consumo. Isso é essencial para plataformas comerciais como a Amazon, onde isso resultaria em um aumento das vendas.

Além disso, a diversidade de itens consumidos também aumenta. SRs permitem que itens não populares sejam encontrados pelos usuários certos, isto é, usuários que estão interessados em consumí-los. Isso só é possível graças a personalização das recomendações, pois caso contrário apenas os itens mais populares seriam sugeridos. Sugerir tais itens de forma aleatória seria contraproducente, uma vez que, por se tratar de itens impopulares, poucos usuários estariam de fato interessados neles. Dessa forma, o catálogo completo de uma plataforma está mais propenso a ser encontrado pelos usuários sem comprometer a qualidade das recomendações, o que é obviamente vantajoso.

Como descrito anteriormente, SRs trazem também inúmeras vantagens para seus usuários. O conjunto de funcionalidades possibilitadas pela implementação de um SR agrega um enorme valor ao produto final e aumenta a satisfação dos usuários. Isso é claramente visto no exemplo do *FilmesJá*. Com o SR implementado, João é não só capaz de encontrar filmes que o interesse com maior facilidade, mas também consegue avaliar filmes, permitindo que ele interaja de uma nova forma com a plataforma. Com um maior nível de satisfação, o uso da plataforma também aumenta.

Quanto mais um usuário usa um SR, mais detalhada fica sua representação no sistema. Isso permite que as recomendações produzidas levem em conta mais características de um usuário e, consequentemente, sejam mais relevantes. Isso faz com que usuários se sintam valorizados ao acessar a plataforma, aumentando sua fidelidade a ela.

Por fim, como SRs monitoram o que é consumido por seus usuários, a plataforma é capaz de extrair uma série de informações valiosas sobre como seu sistema é utilizado. Assim, na hora de definir estratégias de negócio, essas informações podem ser usadas para melhor atender os clientes do sistema. Em aplicações comerciais, por exemplo, pode-se determinar a quantidade de produtos que deve se manter no estoque. Já em plataformas como o *FilmesJá*, é possível identificar o tipo de filme que é mais assistido no site e traçar um plano para incluir mais filmes similares.

Como citado anteriormente, para gerar recomendações, é necessário que SRs armazenem informações sobre seus usuários, suas interações com o sistema e sobre os itens disponíveis. O tipo de informação armazenada varia de acordo com os algoritmos implementados pelo SR. Existem três principais categorias de algoritmos para geração de recomendações, cada uma explorando diferentes aspectos para gerá-las.

A primeira delas, filtragem colaborativa, se baseia na ideia de buscar usuários semelhantes a um usuário específico e criar recomendações a partir dos itens que esse grupo de usuários gostou. Já nas recomendações baseadas em conteúdo, a segunda categoria de algoritmos, o foco está nas características dos itens e dos usuários. Essa classe de algoritmos compara atributos dos itens (preço, sinopse, diretor, por exemplo, no caso de filmes) e dos usuários (idade, gênero, nacionalidade, etc) para identificar semelhanças e gerar recomendações. Por fim, existem as recomendações baseadas em conhecimento. Em geral, estas se baseiam em ontologias para gerar recomendações a partir de preferências explicitas de um usuário.

Antes de melhor detalhar a geração de recomendação usando a filtragem colaborativa, é conveniente que alguns conceitos sejam formalizados. Recomendações baseadas em conhecimento e em conteúdo não serão tratadas nesta monografia e foram citadas apenas por uma questão de completude.

2.1.1 Usuário

Um usuário é uma pessoa para a qual o SR está gerando recomendações. É importante notar que um SR não existe em um vácuo e, portanto, um usuário de um SR é também um

usuário de uma plataforma que esteja usando o SR. Assim, além de receber recomendações, um usuário é também aquele que consome (ou não) os itens recomendados e possivelmente avalia os itens por ele consumidos.

A forma com que um usuário é representado por um SR pode variar de acordo com os algoritmos usados para gerar recomendações. Em alguns casos, salvar atributos demográficos como idade, gênero, ocupação e nacionalidade, pode ser extremamente valioso. Em outros, como no caso da filtragem colaborativa, essas informações podem não ser usadas pelos algoritmos. Nesses casos, usuários são representados de outra forma. Na filtragem colaborativa, por exemplo, usuários são caracterizados apenas por suas transações, como será descrito na seção 2.2.

2.1.2 Item

Um item é aquilo que um SR recomenda e pode variar drasticamente dependendo do domínio do SR. Assim, itens podem variar desde filmes e seriados, como é o caso da *Netflix*, até a livros e outros produtos, como acontece na *Amazon*.

A representação de itens em um SR se dá de forma análoga a maneira como um usuário é representado. Portanto, os algoritmos e informações que serão usadas para gerar recomendações definem como um item será representado. No caso de uma representação por atributos, um filme, por exemplo, teria atributos como título, diretor, atores, ano de lançamento e sinopse para caracterizá-lo. Outra forma de representar um filme seria através dos usuários que o avaliaram, como acontece na filtragem colaborativa.

2.1.3 Transação

Uma transação é definida como uma interação entre um usuário e um item. Os tipos de interações armazenados por um SR depende dos algoritmos que serão usados. Assim, apesar de informações transacionais serem tipicamente compostas por avaliações, elas podem incluir outros tipos de interações. Existem duas categorias de transações que um SR pode explorar: explícitas e implicitas.

Transações explícitas são aquelas em que o usuário ativamente oferece *feedback* sobre um item, seja porque o sistema lhe pediu ou porque ele proativamente decidiu fazê-lo. Avaliações caem dentro dessa categoria e, como mencionado anteriormente, são os tipos de transação mais difundidas na indústria. A forma com que avaliações são representadas também pode variar de acordo com o domínio e os algoritmos usados.

Outro exemplo de transação explícita é permitir que usuários etiquetem itens com termos pré-definidos ou customizáveis. Isso pode levar a uma modelagem mais complexa das preferências de um usuário, já que se tem mais informações sobre suas opiniões.

Já as informações transacionais implicitas são coletadas pelo sistema de forma automática conforme um usuário usa a plataforma. Assim, um SR é capaz de armazenar informações

	Avengers	**Midsommar**	**Joker**	**Bacurau**
João	3	1	5	
Maria	5	5	4	4
Pedro	2	1		3
Ana	4	3	5	3

Tabela 2.1: *Exemplo de matriz de avaliações no domínio de filmes. Note que Pedro não avaliou Joker e João, Bacurau.*

como os itens consumidos e visualizados por um usuário e até, se aplicável, que termos ele usou na busca para encontrar um determinado item. Além disso, em domínios onde essa informação é relevante, como em plataformas de streaming de videos e música, o tempo de consumo também pode ser monitorado.

É importante notar que transações implícitas não são tão confiáveis quanto as explícitas, uma vez que cliques podem não se traduzir em interesse real por um item. Por outro lado, quando um usuário avalia um item de forma explícita, seu interesse (ou desinteresse) pelo item é indiscutível. Ainda assim, a coleta implícita de informações transacionais pode ser muito valiosa em alguns domínios, principalmente quando é possível extrair uma grande quantidade de transações. Isso permite minimizar os efeitos dos casos onde o comportamento do usuário não condiz com sua opinião sobre um item. Além disso, note que, embora as transações explícitas sejam mais confiáveis, elas exigem que usuários estejam dispostos a fornecer suas opiniões para o SR, o que nem sempre acontece.

Devido a dificuldade na obtenção de um grande volume de transações implícitas e pela ausência delas em *datasets* públicos, este trabalho foi desenvolvido utilizando apenas transações explícitas, mais especificamente avaliações. Uma avaliação é uma associação entre um usuário e um item e possui um valor associado a ela. Assim, o conjunto de todas as avaliações pode ser representada, sem perda de generalidade, por uma matriz, onde as linhas representam usuários, as colunas, itens e as interseções entre elas, avaliações, como mostra a tabela 2.1.

2.1.4 Recomendação

Uma recomendação é um conjunto de itens ainda não consumidos por um usuário que o SR julgou ser útil para ele. Recomendações podem vir acompanhadas de uma estimativa de avaliação, mas nem todos os algoritmos calculam essas informações.

A maioria das listas de recomendação é também ordenada, isso é, itens no topo da lista são, teoricamente, de maior interesse para o usuário. Além disso, em alguns casos, é também possível fornecer uma explicação de porque um dado item foi recomendado. Tais explicações podem ser valiosas para convencer os usuários de que o SR usado é de boa qualidade, fazendo com que eles estejam mais dispostos a aceitar as sugestões do recomendador.

2.2 Filtragem Colaborativa

A filtragem colaborativa (FC) é a técnica para geração de recomendações mais difundida e implementada na indústria. Tais algoritmos exploram o fato de que se os usuários u e v se comportaram de forma semelhante no passado, então eles devem se comportar de forma semelhante no futuro. Assim, se u e v avaliaram itens de forma semelhante, então suas avaliações futuras provavelmente também serão parecidas.

Como mencionado nas sessões 2.1.2 e 2.1.1, algoritmos de FC não precisam de informações sobre os itens e usuários para gerar recomendações, já que, em suas formas mais puras, eles utilizam apenas dados transacionais para fazê-lo. Assim, as preferências de um usuário são modeladas a partir de suas interações com os itens de uma plataforma. Isso faz com que não seja necessário que se coloque ou atualize informações sobre itens no sistema, o que pode ser custoso dependendo da quantidade de itens disponíveis.

Portanto, na FC, um usuário pode ser representado apenas como um vetor contendo suas avaliações (linhas da matriz de avaliação). O mesmo acontece para os itens, mas de maneira inversa: cada item pode ser representado por um vetor de notas dadas a ele (colunas da matriz de avaliação). Note que, tanto no caso de novos usuários, quanto no caso de novos itens, esses vetores estariam vazios e, portanto, algoritmos de FC seriam incapazes de gerar recomendações. Para esses casos, outras técnicas e tipos de algoritmos devem ser empregados para gerar recomendações.

De maneira geral, o objetivo de um algoritmo de FC é, dado um usuário e seu vetor de avaliações, estimar um valor para as entradas vazias de seu vetor e devolver os itens com as maiores notas estimadas para aquele usuário. Seja $U = \{u_1, ..., u_n\}$ o conjunto de usuários, $I = \{i_1, ..., i_m\}$ o conjunto de itens e R a matriz de avaliações r_{ij}, com $i \in 1...n$ e $j \in 1...m$. Para cada r_{ij} vazio, um algoritmo de FC seria responsável por calcular a estimativa $\bar{r}_{ij}$ para fornecer recomendações para o usuário i.

Neste trabalho, foram implementados três diferentes algoritmos de FC. Os dois primeiros, filtragem colaborativa baseada em usuário (FCBU) e baseada em itens (FCBI), são os métodos clássicos de geração de recomendação com filtragem colaborativa. Ambos calculam estimativas da avaliação que um usuário daria para um item, mas diferem em como essa estimativa é calculada. Já o terceiro algoritmo, o passeio aleatório (PA), se baseia na construção de um grafo para gerar as recomendações. As sessões a seguir irão detalhar cada um desses algoritmos.

2.2.1 Filtragem Colaborativa Baseada em Usuários

Dado um usuário u e um item i, a FCBU tenta calcular uma estimativa de nota $\hat{r}_{ui}$ que u daria para i com base nos k usuários mais parecidos com u, conhecidos como os k vizinhos mais próximos (kNN). Como o objetivo é calcular essa estimativa para um item específico - e para isso só são levado em conta usuários que avaliaram i - em vez de se utilizar

simplesmente os k vizinhos mais próximos, usa-se os k vizinhos mais próximos que avaliaram i, denotado por $N_i(u)$. O cálculo de $\hat{r}_{ui}$ é dado pela equação 2.1, onde $\bar{r}_u$ representa a nota média dada por u, r_{vi} a nota dada ao item i por v e w_{uv} a similaridade entre u e v (que será discutida na seção 2.3).

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_i(u)} w_{uv} * (r_{vi} - \bar{r}_v)}{\sum_{v \in N_i(u)} |w_{uv}|} \tag{2.1}$$

Assim, para gerar recomendações para o usuário u, este algoritmo olha para seus k vizinhos mais próximos N_u e, para cada vizinho $v \in N_u$, encontra-se a lista $I_v(u)$ de itens avaliados por v que não foram avaliados por u. Para cada $i \in I_v(u)$ calcula-se, então, uma estimativa de avaliação. Por fim, o algoritmo devolve a lista dos n itens com maior estimativa, ordenada por valor decrescente de $\hat{r}_{ui}$.

Algorithm 1: Filtragem Colaborativa Baseada em Usuários

```
u ← currentUser();
N ← similarUsers(u);
recommendations ← empty list;
for each item in itemsRatedBy(N, k) \ itemsRatedBy(u) do
    ratingPrediction ← calculatePredictedRating(u, item);
    recommendations.push([item, ratingPrediction]);
end for
return pickBestItems(recommendations, n)
```

2.2.2 Filtragem Colaborativa Baseada em Itens

A FCBI é análoga ao algoritmo anterior, mas ao invés de olhar para os usuários vizinhos de u, olha para os items que u avaliou. Seja $N_u(i)$ o conjunto dos k itens avaliados por u mais parecidos com i. A estimativa de nota que u daria para um item i pode ser calculada de acordo com a equação 2.2, onde w_{ij} é a similaridade entre os itens i e j e $r_{u,i}$ é a nota que u deu para i.

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u(i)} w_{ij} * r_{uj}}{\sum_{j \in N_u(i)} |w_{ij}|} \tag{2.2}$$

Ou seja, para gerar recomendações para um usuário u, para cada item i avaliado por u, encontra-se $I_u(i)$, o conjunto de items similares a i que u não avaliou. Para cada $i \in I_u(i)$, calcula-se $\hat{r}_{ui}$. O algoritmo devolve, então, a lista dos n items mais relevantes e suas respectivas estimativas.

Algorithm 2: Filtragem Colaborativa Baseada em Items

```
u ← currentUser();
I ← itemsRatedBy(u);
recommendations ← empty list;
for each item in similarItems(I, k) \ I do
        ratingPrediction ← calculatePredictedRating(u, item);
        recommendations.push([item, ratingPrediction]);
end for
return pickBestItems(recommendations, n)
```

2.2.3 Passeio Aleatório

Como dito anteriormente, este algoritmo é baseado em grafos. Portanto, antes de descrevê-lo é importante que seja definido como esse grafo é montado. O grafo é composto pelas três principais entidades em um SR: usuários, itens e transações. Usuários e itens são considerados os nós e as transações, as arestas. Assim, um usuário está conectado a todos os itens que avaliou. Além disso, existe um segundo tipo de aresta: as arestas de similaridade. Diferente das arestas de avaliação, arestas de similaridade conectam apenas nós do mesmo tipo, ou seja, usuários com usuários e itens com itens. Dois nós são conectados por uma aresta de similaridade se a similaridade entre eles é maior do que um determinado limite.

Com o grafo devidamente descrito, é possível agora descrever o algoritmo do PA. A partir do nó que representa o usuário u, são feitos p passeios aleatórios de tamanho l, sendo possível atravessar qualquer tipo de aresta (similaridades e avaliações) em qualquer direção. No último passo do passeio, o algoritmo é obrigado a escolher um nó de filme, escolhendo arestas de similaridade se estiver em um filme ou uma aresta de avaliação se estiver em um usuário. Os filmes atingidos pelo algoritmo são computabilizados e uma lista dos n filmes mais atingidos é devolvida.

Algorithm 3: Passeio Aleatório

```
u ← currentUserNode();
recommendations ← empty list;
for i in 1..p do
        item ← randomWalk(u, l);
        recommendations.push(item);
end for
return pickMostVisitedItems(recommendations, n)
```

Uma variante desse algoritmo também foi implementada, onde a escolha das arestas é feita de forma enviesada, favorecendo aquelas de maior peso. O peso de uma aresta, por sua vez, é definido pela similaridade, se for uma aresta de similaridade, ou pela nota dada, caso seja uma aresta de avaliação. Tal variação do algoritmo é chamada de Passeio Aleatório Enviesado (PAE).

Ambos os algoritmos de passeio aleatório são considerados algoritmos híbridos de grafos,

uma vez que eles não se restringem a usar apenas alguns tipos de arestas, podendo explorar informações tanto de itens quanto de usuários para gerar as recomendações.

2.3 Medidas de similaridade

Como é possível perceber, todos os algoritmos descritos na seção 2.2 dependem de determinar a similaridade entre itens e usuários. Para isso, são utilizadas medidas de similaridade: funções reais usadas para determinar quão parecidos dois objetos são. Para isso, objetos são representados como vetores de n dimensões, onde cada dimensão representa um de seus atributos.

A escolha de quantos e quais atributos serão utilizados pela medida de semelhança depende do domínio em que se está sendo aplicado e influencia fortemente o resultado. Assim, é de extrema importância que se tome cuidado na escolha de quais atributos serão utilizados. Além disso, o cálculo dessas medidas de similaridade podem ser extremamente custoso para bases grandes, já que o número de operações cresce de forma quadrática com o número de usuários e itens.

No caso da FC, esses atributos são, na verdade, os vetores de avaliações, conforme mencionado no início da seção 2.2. Dessa forma, um usuário é representado por sua respectiva linha na matriz de avaliação, e um item por sua respectiva coluna.

Existem várias formas de se calcular a similaridade entre dois objetos. A seguir serão descritas duas maneiras diferentes de fazê-lo, usando a similaridade de cossenos e a similaridade de pearson. Essas diferentes medidas de similaridade serão usadas e seus impactos sobre a geração de recomendações serão discutidos nos resultados dos experimentos descritos no capítulo 3.

2.3.1 Similaridade de Cossenos

Similaridade de cossenos é uma medida de similaridade baseada no ângulo entre dois vetores. Dois vetores são considerados mais similares se suas orientações são parecidas, independentemente de seus tamanhos. Assim, se o ângulo θ entre eles é de $0°$, suas similaridades são máximas, valendo 1, enquanto se eles estão em direções opostas ($\theta = 180°$), suas similaridades são mínimas, valendo -1. A similaridade de cossenos é definida, portanto, como:

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{2.3}$$

Trazendo a equação 2.3 para o contexto de SR, a similaridade de cossenos pode ser reescrita de acordo com equação 2.4, onde u e v são usuários, I_{uv} são os itens avaliados por

u e v, r_{ui} a avaliação de i por u e I_u os itens avaliados por u.

$$w_{uv} = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2 \sum_{i \in I_v} r_{vi}^2}} \tag{2.4}$$

O cálculo da similaridade entre itens, por sua vez, é feito de forma análoga a dos usuários, como mostra a equação 2.5.

$$w_{ij} = \frac{\sum_{u \in U_{ij}} r_{ui} r_{uj}}{\sqrt{\sum_{u \in U_i} r_{ui}^2 \sum_{u \in U_j} r_{uj}^2}} \tag{2.5}$$

2.3.2 Coeficiente de Pearson

O Coeficiente de Pearson é uma medida da correlação linear entre duas variáveis. Seu valor varia de -1 a 1, onde -1 representa uma correlação linear negativa, 0 a ausência de uma correlação linear e 1 uma correlação linear positiva. A fórmula para calcular o coeficiente de Pearson é:

$$\rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.6}$$

Assim, no contexto de sistemas de recomendação, pode-se definir a similaridade de coeficiente de pearson entre os usuários u e v de acordo com a equação 2.7, onde I_{uv} representa o conjunto de itens avaliados por u e v, r_{ui} a nota dada por u ao item i e $\bar{r}_u$ a nota média dada por u.

$$w_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2}\sqrt{\sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}} \tag{2.7}$$

A equação , análoga a equação 2.7, representa a similaridade entre os itens i e j. Seja U_{ij} o conjunto de usuários que avaliou os itens i e j, $\bar{r}_i$ a nota média de i, e $r_u i$ a nota dada por um usuário u a i.

$$w_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)^2}\sqrt{\sum_{u \in U_{ij}} (r_{uj} - \bar{r}_j)^2}} \tag{2.8}$$

2.4 Avaliando Sistemas de Recomendação

Ao se implementar um sistema de recomendação, existem muitas formas de avaliar e testar os algoritmos implementados. Em sistemas "reais", por exemplo, pode-se contar contar

com o feedback dos usuários para se determinar se as recomendações sendo feitas estão boas ou não. Já com dados históricos, como é o caso deste trabalho, não é possível determinar se um usuário gostou ou não de uma recomendação se ele nunca avaliou o item em questão.

Uma das formas mais comuns de se trabalhar com dados históricos é particionar o dataset em duas partes: uma partição de treinamento e uma de testes [5]. Como sugere o nome, a partição de treinamento é responsável por treinar o algoritmo, isto é, fornecer dados de base para que o algoritmo possa funcionar de forma correta. O segundo grupo de usuários, por sua vez, é responsável por avaliar a performance do algoritmo. Para isso, apenas uma parte de suas avaliações são incluídas, fazendo com que as excluidas possam ser usadas para avaliar os algoritmos.

2.4.1 Métricas

Para poder avaliar e comparar os algoritmos implementados, é importante definir as métricas que serão utilizadas. Herlocker et al. [3] define algumas das principais métricas usadas para avaliar sistemas de recomendação e algumas delas foram escolhidas para este trabalho.

A primeira delas, o erro médio absoluto, está relacionada a avaliação da precisão de algoritmos que tentam estimar a nota do usuário com relação aos itens. Tais métricas são importantes para que se possa avaliar a qualidade das estimativas feitas. Quanto melhor as estimativas, melhor serão as recomendações feitas, já que será possível criar uma lista de itens que realmente interesse o usuário. A equação 2.9 descreve o cálculo do erro médio absoluto, onde r_{ui} representa a nota dada por u para o item i, $\hat{r}_{ui}$ a nota estimada pelo recomendador e R_{test} o conjunto de recomendações da partição de teste.

$$\text{MAE} = \frac{\sum_{r_{ui} \in R_{\text{test}}} |\hat{r}_{ui} - r_{ui}|}{|R_{\text{test}}|} \tag{2.9}$$

Note que, embora seja muito útil para validar a qualidade das recomendações cujo valor $r_{u,i}$ é sabido, o erro médio absoluto, não mede a relevância das outras recomendações geradas. Essa é uma distinção importante de ser feita quando se trabalha com dados históricos pois não há como inferir se um usuário gostaria ou não da recomendação feita se ele nunca avaliou um item. Assim, mesmo com um valor baixo de MAE, é possível que um conjunto de recomendações ainda seja irrelevante para um usuário.

Por exemplo, suponha que um usuário u receba como recomendação o conjunto de filmes $M = m_0, m_1, ...m_n$ e que u só tenha avaliado o filme m_0. Se $pred(u, m_0)$ for próximo a r_{u,m_0}, então o valor do MAE será baixo. Como não se sabe como u avaliaria os filmes $m_1...m_n$, pode ser que eles sejam irrelevantes para o usuário e a medida de erro médio absoluto não tem como indicar isso.

Para que isso possa ser enxergado, são úteis as métricas de precisão e *recall*. Enquanto a precisão representa a probabilidade de um item recomendado ser relevante, o *recall* repre-

senta a probabilidade de um item relevante ser selecionado.

A precisão é definida como a razão entre o número de itens recomendados pelo algoritmo que o usuário realmente avaliou e gostou ($|H_u|$) e o número total de itens recomendados para um usuário ($|R_u|$), como mostra a equação 2.10. O recall, por sua vez, é definido pela razão entre $|H_u|$ e o número de itens que o usuário avaliou e que foram usados para treinar os algoritmos ($|T_u|$). O cálculo do *recall* é feito, portanto, de acordo com a equação 2.11.

$$\mathcal{P}_u = \frac{|H_u|}{|R_u|} \tag{2.10}$$

$$\mathcal{R}_u = \frac{|H_u|}{|T_u|} \tag{2.11}$$

É importante notar que aumentar ou diminuir o número de recomendações feitas para um usuário pode variar drasticamente o valor do *recall* e da precisão. Quanto mais itens forem recomendados, maior será o *recall* e menor será a precisão. Se menos items forem recomendados, o oposto acontece: o *recall* diminui e a precisão aumenta. Isso significa que essas medidas são inversamente relacionadas. Assim, para que elas sejam combinadas em uma só, pode-se utilizar a medida F1, definida pela equação 2.12.

$$F1_u = \frac{2 \cdot \mathcal{P}_u \cdot \mathcal{R}_u}{\mathcal{P}_u + \mathcal{R}_u} \tag{2.12}$$

Capítulo 3

Implementação

3.1 Dados Usados

Para a elaboração deste trabalho, foi utilizado o conjunto de dados da plataforma MovieLens, disponibilizado por *GroupLens Research* [2]. A base é composta por 100000 avaliações de 1682 filmes feitas por 943 usuários e é disponibilizada em formato CSV.

Nela, usuários são representados apenas por um id e, exceto por suas avaliações, não há nenhum outro tipo de informação sobre eles. Já os filmes, contam com mais informações: além de incluir seus gêneros, títulos e ano de lançamento, a base tem também seus respectivos ids no IMDb [4] e no TMDb [11]. As tabelas 3.1 e 3.2 ilustram como esses dados são representados pelos arquivos CSV.

movieId	**title**	**genres**
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy

Tabela 3.1: *Exemplos de entrada no arquivo movies.csv, que contém a lista de filmes na base.*

movieId	**imdbId**	**tmdbId**
1	0114709	862
2	0113497	8844

Tabela 3.2: *Exemplos de entrada no arquivo links.csv, que contém a relação entre os ids da base, do IMDb e do TMDb.*

As avaliações, por sua vez, são compostas apenas pela nota dada pelo usuário e por um *timestamp* do momento da avaliação, como mostra a tabela 3.3. As avaliações são feitas em uma escala de 0.5 a 5.0 com granularidade de 0.5 e notas mais altas representam maior interesse por um filme.

Com o intuito de tornar a implementação mais próxima de um cenário real, facilitar a consulta e garantir a persistência dos dados, foi decidido armazenar os dados em um banco

userId	movieId	rating	timestamp
1	2	3.5	1112486027
1	29	3.5	1112484676

Tabela 3.3: *Exemplos de entrada no arquivo ratings.csv, que contém as avaliações feitas.*

de dados. Assim, os arquivos CSV contendo os dados foram pré-processados e inseridos em um banco de dados Neo4j.

3.2 Neo4j

Neo4j [6] é um banco de dados orientado a grafos altamente escalável. Por ser orientado a grafos, os dados são representados através de nós (entidades) e arestas (relacionamento entre elas). Tanto nós quanto arestas podem ter propriedades. Além disso, arestas sempre têm direção, ou seja, trata-se de um grafo direcionado. Apesar disso, Cypher, a linguagem de consulta do Neo4j, permite tratá-lo como um grafo não direcionado, percorrendo relacionamentos em qualquer direção.

Por tratar-se de um banco de dados não relacional, o Neo4j apresenta também algumas vantagens sobre bancos de dados tradicionais. Primeiramente, existe uma grande flexibilidade na modelagem dos dados, sendo ele facilmente adaptável a novas demandas do sistema. Isso permite que a modelagem dos dados seja muito mais simples, uma vez que não é necessário tentar prever futuras demandas. Além disso, há ganhos em perfomance, já que, ao contrário de bancos de dados relacionais, o número e profundidade das relações não impacta tanto seu desempenho.

O Neo4j permite também a instalação de plugins que adicionam rotinas e novas funcionalidades ao BD. Uma dessas bibliotecas, *Graph Algorithms*, desenvolvida pelo projeto *Neo4j Labs* do próprio Neo4j, implementa não só algoritmos para percorrer grafos, como passeios aleatórios e buscas em largura e profundidade, mas também algoritmos para calcular semelhanças entre nós. Assim, o uso dessas bibliotecas facilitou o cálculo das similaridades entre os usuários e os filmes e, como os dados já estavam representados na forma de um grafo, o desenvolvimento dos algoritmos de passeio aleatório foram ainda mais simples.

3.3 Representação dos Dados

Antes de inserir os dados no banco, foi necessário extrair os dados dos arquivos CSV e pré-processá-los para facilitar sua inserção no Neo4j. Para isso, foram escritos dois scripts em Node.js [8].

O primeiro deles, responsável por gerar um CSV com os usuários a serem inseridos no banco, lê o arquivo com as avaliações feitas e extrai os *userIds* existentes. O arquivo de saída é, portanto, um CSV com todos os *userIds*, sem repetições.

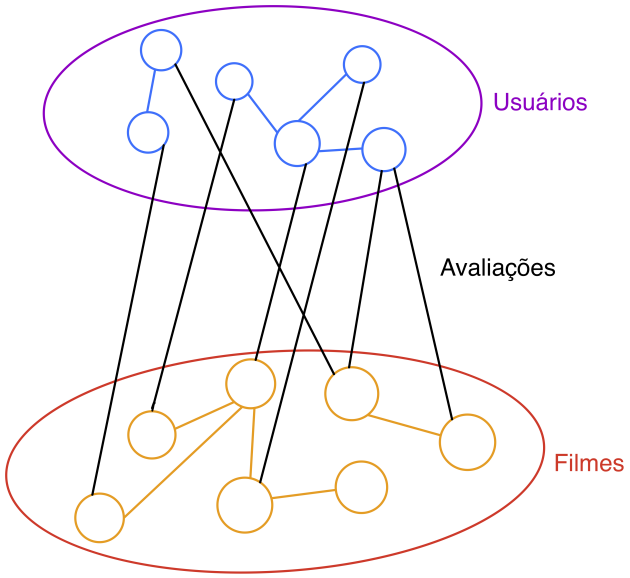

Figura 3.1: *Representação dos grafos gerados. Note que avaliações conectam usuários a filmes, enquanto as medidas de similaridade os conectam entre si.*

O outro script, por sua vez, é responsável pelos filmes. Assim, o script lê o arquivo de filmes e extrai os anos de lançamento de seus títulos. Assim, o CSV de saída desse script consiste em uma lista de filmes com título, gênero e ano de lançamento.

Com esses dados pré-processados, a inserção deles foi feita usando um script em Cypher, que além de inserir os filmes e usuários dos CSVs gerados, cria restrições de unicidade sobre seus ids. Isso não só garante a integridade dos dados originais, mas também automaticamente cria índices no banco para que consultas pudessem ser feitas usando os ids originais de forma eficiente. O script lê também o arquivo de avaliações para criar relações entre filmes e usuários.

Nesse primeiro momento, o banco de dados consiste em um grafo bipartido, onde os nós representam usuários e filmes e as arestas, as avaliações. Note que trata-se de um grafo bipartido pois não existem arestas nem entre filmes, nem entre usuários. Durante a execução dos experimentos, no entanto, arestas de similaridade serão criadas entre usuários e entre itens, formando um grafo semelhante ao da figura 3.3.

É importante notar que, apesar de terem sido inseridas no banco de dados algumas informações como título, gênero e ano de lançamento, esses atributos não são explorados de maneira alguma pelos algoritmos implementados e foram incluídos apenas para facilitar a identificação dos filmes ao explorar a interface gráfica do Neo4j. As informações usadas pelos algoritmos estão restritas apenas às avaliações feitas pelos usuários. Ou seja, do ponto de vista dos algoritmos, usuários e itens são representados por seus vetores na matriz de avaliações.

3.3.1 Implementação dos Algoritmos

Os algoritmos descritos na seção 2.2 foram implementados em Node.js usando a biblioteca neo4j-driver para se conectar com o Neo4j e executar queries. Dessa forma, os algoritmos foram implementados com uma combinação de funções em Node.js e queries em Cypher.

A fim de diminuir o número de chamadas ao banco, sempre que possível, as queries foram implementadas para executar comandos em batchs. Assim, o overhead de acesso ao banco de dados é reduzido, aumentando a velocidade com que consultas podem ser feitas. As queries nas listagens 3.1 e 3.2 são um exemplo disso, já que gera recomendações para um grupo de usuário ao invés de fazê-lo individualmente.

```
MATCH (u:User)-[sim:${userRelationship}]-(v:User)
MATCH (v)-[r:RATES]->(m:Movie)
WHERE NOT (u)-[:RATES]->(m) AND u.movieLensId IN [${userIds}]
WITH u, v, m, sim, r
ORDER BY sim.similarity DESC
WITH u, m, collect({normalizedRating: r.rating - v.avgRating,
    similarity: sim.similarity})[..${k}] as similarUsers
WITH u, m, u.avgRating +
    reduce(num = 0, v in similarUsers |
        num + v.similarity * v.normalizedRating) /
    reduce(den = 0, v in similarUsers |
        den + abs(v.similarity)) as score
ORDER BY score DESC
WITH u, collect({m: m.movieLensId, score: score})[..${n}]
    as recommendations
UNWIND recommendations as rec
MATCH (m:Movie {movieLensId: rec.m})
MERGE (u)-[r:PROBABLY_LIKES_UB {score: rec.score}]->(m)
```

Listing 3.1: *Geração de recomendações baseadas em usuário usando Cypher para todos os usuários em uma lista (userIds).*

É importante notar que em ambas as listagens algumas variáveis (userIds, movieRelationship, userRelationship etc.) estão sendo dinamicamente colocadas na query. Isso só é possível devido ao fato da query estar sendo montada em tempo de execução no programa em Node.js.

```
MATCH (u:User)-[r:RATES]->(m:Movie)
MATCH (m)-[sim:${movieRelationship}]-(similarMovies:Movie)
WHERE NOT (u)-[:RATES]->(similarMovies)
    AND u.movieLensId IN [${userIds}]
WITH u, similarMovies, r, sim
```

```
    ORDER BY sim.similarity DESC
    WITH u, similarMovies, collect(
            {rating: r.rating, similarity: sim.similarity}
        )[..${k}] as ratings
    WITH u, similarMovies,
        reduce(num = 0, rating in ratings |
            num + rating.similarity * rating.rating) as num,
        reduce(den = 0, rating in ratings |
            den + abs(rating.similarity)) as den
    WITH u, similarMovies, num / den as score
    ORDER BY score DESC
    WITH u, collect(
        {m: similarMovies.movieLensId, score: score}
    )[..${n}] as recommendations
    UNWIND recommendations as rec
    MATCH (m:Movie {movieLensId: rec.m})
    MERGE (u)-[r:PROBABLY_LIKES_IB {score: rec.score}]->(m)
```

Listing 3.2: *Geração de recomendações baseadas em items usando Cypher para todos os usuários em uma lista (userIds).*

Como pode-se perceber nas listagens 3.1 e 3.2, toda vez que uma recomendação é gerada, uma nova aresta é criada no banco representando essa recomendação. Tais arestas são usadas posteriormente para que sejam calculadas as métricas dos experimentos. Além disso, isso permite uma fácil visualização das recomendações criadas, uma vez que pode-se utilizar a interface gráfica do Neo4j para visualizar o grafo gerado.

Existem algums detalhes de implementação que valem a pena ser mencionados. Foram empregadas algumas técnicas de redução de tamanho de vizinhança de filmes e usuários. A primeira delas, já mencionada na seção 2.2, consiste em usar apenas os k vizinhos mais parecidos para o cálculo das estimativas de nota. Isso não só garante que o algoritmo possa ser rodado mais rapidamente, mas também diminui o ruído trazido por vizinhos menos relevantes. O valor de k foi definido empiricamente como 40.

A segunda técnica é o uso de um limite t de similaridade para que dois objetos sejam de fato considerados similares. Ou seja, dois nós só são conectados por uma aresta de similaridade se $w_{nm} \geq t$. Isso é especialmente importante para os algoritmos de passeio aleatório, já que os algoritmos FCBU e FCBI já são limitados pelo valor de k. No caso dos passeios aleatórios, esse limite diminui drasticamente o tamanho das vizinhanças dos nós, fazendo com que os algoritmos consigam gerar recomendações de maior qualidade, já que tendem a escolher nós mais similares.

3.4 Descrição dos Experimentos

O principal objetivo dos experimentos é conseguir avaliar os algoritmos de recomendação implementados e identificar a influência de certos parâmetros sobre a qualidade das recomendações. Para compará-los utilizou-se as métricas descritas na seção 2.4.1. Dessa forma, foi escrito um script em Node.js que, para cada cenário de teste, prepara a base de dados e roda os experimentos.

O preparo da banco de dados envolve algumas etapas. Antes de mais nada, é necessário limpá-lo, o que significa apagar quaisquer recomendações, resetar modificações feitas em usuários e avaliações e, por fim, apagar as arestas de similaridade.

Então, como mencionado na seção 2.4, os dados são particionados em duas partes: uma que será usada para fornecer dados ao modelo e outra que será usada para os testes. Para garantir que os resultados dos testes sejam consistentes, é usada a técnica de k-folds com $k = 5$, onde o dataset é particionado em k pedaços e, para cada um deles, os testes são rodados com aquela partição como a de testes. Assim, cada fold é composto por dois conjuntos: um de tamanho $\frac{1}{k}$ do total de avaliações, denominado partição de testes, e outro de tamanho $\frac{k-1}{k}$, denominado partição de treinamento. A partição de testes é excluída da fase de treinamento dos algoritmos, que nesse caso, representa o cálculo das similaridades. Isso significa que, como $k = 5$, 20% das avaliações são desabilitadas de forma que os algoritmos não as levem em consideração na hora de gerar recomendações ou calcular similaridades. Na prática, arestas de avaliação (RATES) são transformadas em arestas de avaliação desabilitadas (DISABLED_RATES) para que elas não sejam computadas pelos algoritmos. Note que foi escolhido apenas renomear tais arestas porque deletá-las seria inconveniente para resetar o banco ao seu estado inicial para cada fold.

Assim, com as avaliações da partição de teste devidamente ignoradas, são feitos os cálculos das similaridades entre os filmes e usuários para que eles possam ser utilizados pelos algoritmos. As similaridades foram calculadas de acordo com os procedimentos descritos na seção 2.3. Como são calculados dois tipos de similaridade, dois tipos de aresta são criados (COS_SIM e PEARS_SIM) e cada cenário de teste é responsável por utilizar apenas os tipos de arestas adequados para o cenário.

Então, com as similaridades em mãos, é possível gerar recomendações coms os diferentes algoritmos. Assim, para cada usuário, cada algoritmo gera suas recomendações e as salvam no banco. Por fim, as métricas descritas nas seção 2.4.1 são calculadas para esse fold. Os experimentos são então repetidos com os folds seguintes. Os resultados de cada fold são computados e uma média de cada uma das métricas é calculada para compor os resultados finais dos experimentos.

Para o cálculo das métricas, como as arestas de recomendação foram salvas no banco, é fácil de avaliar a qualidade de uma recomendação. Para o cálculo do MAE, basta comparar os valores das arestas de avaliações desabilitadas com as arestas de recomendação. Já para o cálculo das métricas de precisão, recall e F1, basta identificar cenários em que a recomen-

dação foi um verdadeiro positivo, falso negativo e verdadeiro negativo. A figura ilustra esses cenários. Note que as arestas de recomendação são do tipo PROBABLY_LIKES.

Figura 3.2: *Verdadeiro positivo*

Figura 3.3: *Verdadeiro negativo*

Figura 3.4: *Falso negativo*

Figura 3.5: *Exemplos de recomendações geradas pelo algoritmo FCBI consideradas verdadeiro positivo, falso negativo e verdadeiro negativo*

Foram criados alguns cenários de teste diferentes para os diferentes algoritmos. Variou-se o tamanho l dos passeios aleatórios para testar a sua influência sobre a qualidade das recomendações dos algoritmos de passeio aleatório. Além disso, o número n de recomendações feitas também foi variado para que se avalie sua influência sobre as métricas usadas. Por fim, as medidas de similaridade em si foram testadas. As tabelas 3.4 mostram os cenários de teste criados. Note que os cenários descritos não são válidos para alguns algoritmos. Por exemplo, apenas os algoritmos de passeio aleatório podem variar o parâmetro l e a medida de similaridade entre filmes não é relevante para a FCBU.

Parâmetro		**Valores**
Similaridade	**Filmes**	[Pearson, Cosseno]
	Usuários	[Pearson, Cosseno]
l		[3, 5]
n		[25, 50, 100]

Tabela 3.4: *Variações dos cenários de teste para os algoritmos implementados*

Capítulo 4

Resultados

A tabela 4.1 mostra os melhores resultados obtidos por cada um dos algoritmos e a configuração correspondente. É possível observar que os algoritmos de passeio aleatório tiveram um desempenho consideravelmente pior do que os outros dois algoritmos. Os erros médios absolutos, por sua vez, se mostraram baixos, indicando que as estimativas feitas e "acertadas"foram boas. A seguir serão discutidos alguns outros resultados interessantes que puderam ser observados.

Primeiramente, é interessante notar que, com exceção da FCBI, todos os algoritmos tiveram sua melhor performance com altos valores de n. A figura 4.1 mostra como conforme n aumenta, a performance dos algoritmos clássicos de FC também aumenta, com exceção da FCBI com a similaridade de Pearson. Isso indica que, na maior parte dos casos, os ganhos nos valores de recall compensam os valores perdidos de precisão.

Do ponto de vista dos algoritmos de passeio aleatório, o valor de n também parece ter um impacto positivo sobre a performance dos algoritmos, como mostra a figura 4.2. O mesmo comportamento pode ser observado para o algoritmo PAE.

É importante notar que, apesar de altos valores de n apresentarem melhores resultados de acordo com as métricas utilizadas, isso pode não se traduzir em melhorias verdadeiras em sistemas reais, onde o usuário não está disposto a olhar grandes listas de recomendações. Assim, nesse sentido, listas com menos filmes podem ser mais apreciadas pelo usuário final.

	FCBU	**FCBI**	**PA**	**PAE**
Similaridade - Usuários	Pearson	-	Cosseno	Pearson
Similaridade - Filmes	-	Pearson	Cosseno	Cosseno
n	100	50	100	100
l	-	-	3	3
P	5,74%	6,37%	3,41%	3,41%
R	32,33%	19,56%	29,66%	28,35%
F1	9,16%	8,91%	5,59%	5,57%
MAE	0,7580	0,6967	-	-

Tabela 4.1: *Melhores resultados apresentados pelos algoritmos com os respectivos cenários.*

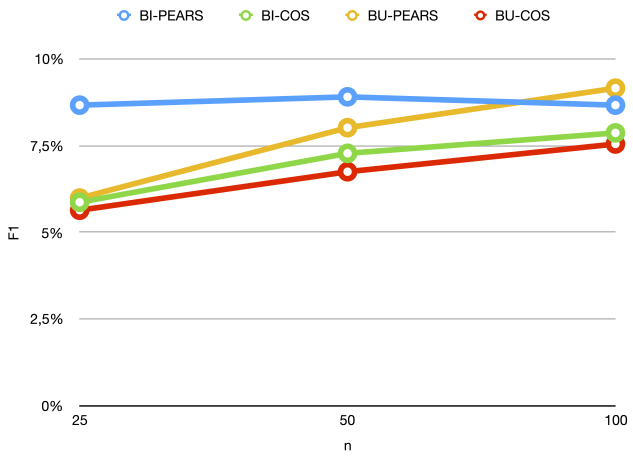

Figura 4.1: *Impacto do número de recomendações sobre F1 para os algoritmos FCBU e FCBI*

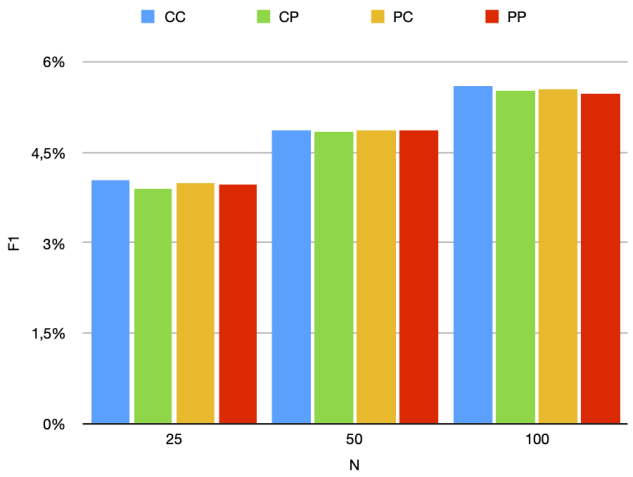

Figura 4.2: *Impacto do número de recomendações sobre F1 para os algoritmos PA com l=3*

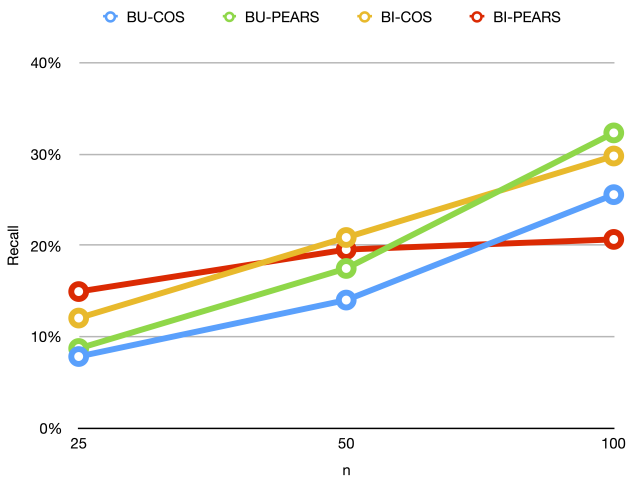

Figura 4.3: *Impacto do número de recomendações sobre o recall para os algoritmos FCBU e FCBI*

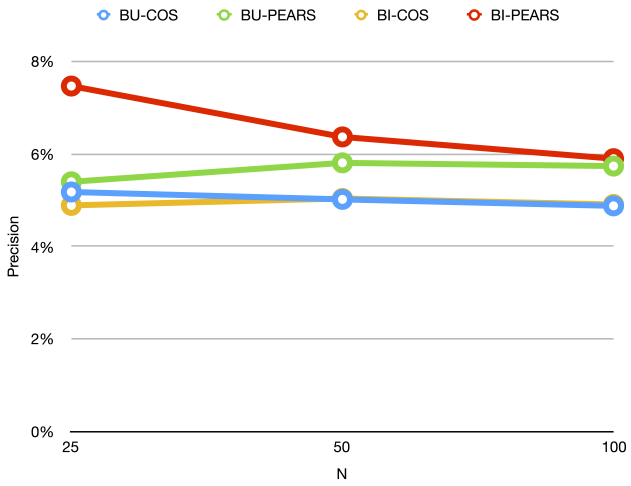

Figura 4.4: *Impacto do número de recomendações sobre a precisão para os algoritmos FCBU e FCBI*

As figuras 4.3 e 4.4 mostram como o número de recomendações afeta, em geral, positivamente os valores de recall e negativamente os de precisão. Além disso, é interessante notar como o recall é mais sensível ao valor de n do que a precisão. Isso indica que boa parte das recomendações novas são, de fato, úteis pois, caso contrário, a quantidade de falso positivos aumentaria muito, diminuindo o valor da precisão ainda mais. Ainda assim, encontrar o balanço entre o valor de recall e precisão ideais não é uma tarefa simples.

O impacto de n sobre o erro médio absoluto, por sua vez, é razoavelmente pequeno, como indica o gráfico na figura 4.5. Pode-se perceber que o valor do MAE tende a cair levemente conforme n aumenta, com exceção do caso da FCBI com similaridade de Pearson. A figura também sugere que as diferentes medidas de similaridade são determinantes em definir o MAE. No caso da FCBU, por exemplo, apesar da diferença entre os MAEs das Similaridades de Cosseno e de Pearson não ser tão grande, ela é quase constante para os diferentes valores de n. Além disso, é interessante destacar que a Similaridade de Pearson

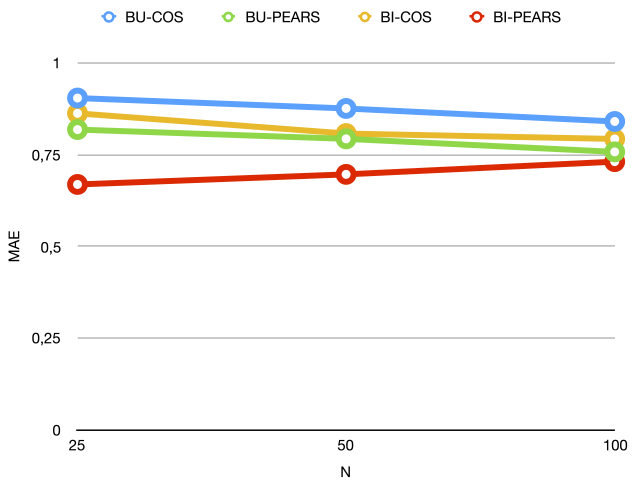

Figura 4.5: *Impacto do número de recomendações sobre o erro médio absoluto para os algoritmos FCBU e FCBI*

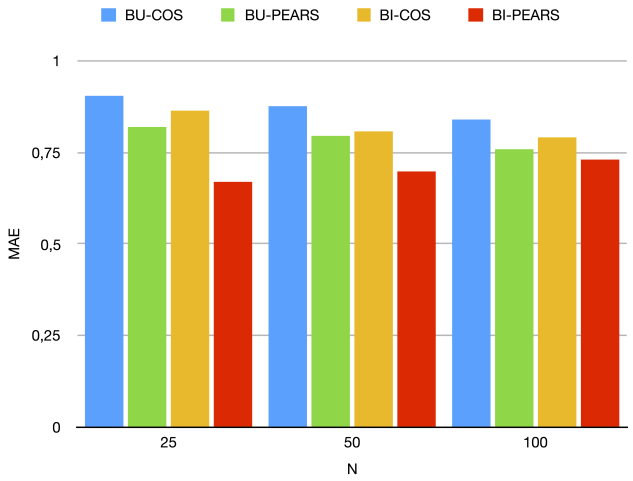

Figura 4.6: *Impacto das medidas de similaridade sobre o MAE para os diferentes valores de n*

garantiu MAEs menores em ambos os algoritmos, como mostra a figura 4.6.

No caso dos passeios aleatórios, a escolha da medida de similaridade parece ter um impacto ainda menor sobre a acurácia das recomendações. A figura 4.7 mostra como as diferentes medidas de similaridade tem um baixíssimo efeito sobre o valor de F1. Apesar da figura mostrar apenas para o caso onde n = 100, esse mesmo comportamento pode ser observado para outros valores de n.

A figura 4.7 revela também que o tamanho dos passeios aleatórios tem um impacto negativo na qualidade das recomendações geradas, tanto no caso do PA quanto no caso do PAE. Além disso, curiosamente, o algoritmo enviesado parece ter uma desvantagens sobre o passeio aleatório não enviesado, o que talvez possa ser explicado por duas decisões de implementação.

A primeira delas, está relacionada a redução do número de vizinhos com o uso de limites para as similaridades. Ao restringir relacionamentos apenas a nós muito similares, o PAE

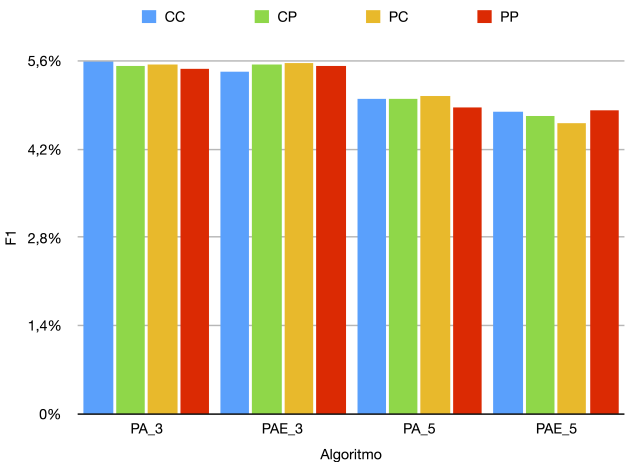

Figura 4.7: *Impacto das medidas de similaridade sobre a performance dos algoritmos de passeio aleatório com n = 100. Note como a maior variação entre os valores de F1 é de apenas 0,21%.*

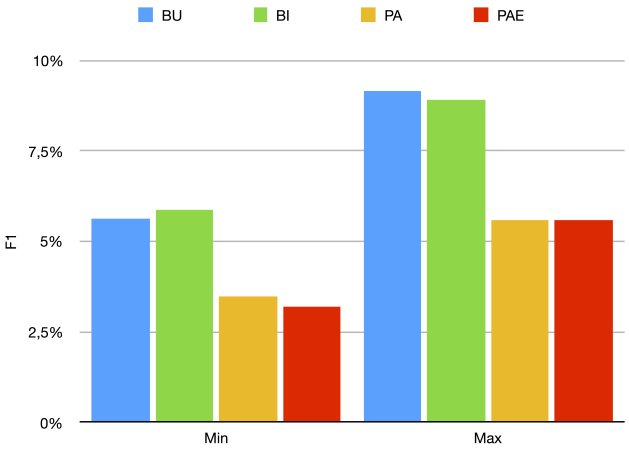

Figura 4.8: *Comparação entre os piores e melhores medidas F1 dos algoritmos testados: (1) BU - pior: COS n=25, melhor: PEARS n=100; (2) BI - pior: COS n=25, melhor: PEARS n=50; (3) PA - pior: l=5 u=PEARS m=PEARS n=25, melhor: l=3 u=COS m=COS n=100; (4) PAE - pior: l=5 u=PEARS m=COS n=25, melhor: l=3 u=PEARS m=COS n=100*

acaba perdendo uma de suas grandes vantagens: evitar arestas com pesos muito baixos.

A segunda decisão que pode ter impactado esses resultados está no fato de que foram feitas apenas 100 passeios por recomendação. Como o dataset é um grafo bastante denso, talvez a quantidade de passeios não esteja suficientemente grande para que nós sejam visitados mais de uma vez, fazendo com que o algoritmo enviesado não tenha tantas oportunidades de fazer boas escolhas.

Ao se comparar os resultados dos algoritmos de passeio aleatório com os algoritmos clássicos de filtragem colaborativa é possível perceber que a FCBU e FCBI se mostraram consistentemente melhores do que o PA e o PAE. De fato, ao se comparar os melhores resultados dos passeios aleatórios com os piores resultados dos outros dois algoritmos, percebe-se que ainda assim a FCBU e FCBI ganham. A figura 4.8 mostra justamente esse cenário.

Por fim, comparando a performance do FCBU e FCBI entre si, percebe-se que a FCBI

tem uma leve vantagem na maioria dos cenários, tanto nos valores de MAE quanto nos valores de F1. Isso é condizente com os resultados obtidos por [9], mas pode também ser uma característica da base de dados utilizada, que contém uma quantidade de itens maior do que a de usuários.

Capítulo 5

Conclusão

A implementação de diferentes algoritmos com diferentes parâmetros reiteram a complexidade do desenvolvimento de um bom sistema de recomendações na medida em que encontrar as melhores configurações para o sistema provou ser uma tarefa extremamente complexa, ainda mais quando vários algoritmos são utilizados. Além disso, encontrar um balanço entre as métricas também se mostrou desafiador.

Os algoritmos de filtragem colaborativa baseada em itens e baseada em usuários tiveram desempenhos semelhantes, mas o primeiro gerou recomendações levemente mais relevantes de acordo com as métricas utilizadas. Porém, ambos os algoritmos tiveram uma performance consideravelmente melhor do que os algoritmos de passeios aleatórios. O PAE, por sua vez, teve, surpreendentemente, o pior desempenho dentre os algoritmos testados.

O uso das diferentes medidas de similaridade mostraram como elas podem afetar a qualidade das recomendações, principalmente no caso dos algoritmos de FCBU e FCBI. De forma geral, percebeu-se que o uso da similaridade de Pearson se mostrou vantajosa para ambos os algoritmos. Já no caso dos algoritmos de passeio aleatório, as diferentes medidas de similaridade tiveram pouco impacto nos resultados.

Em trabalhos futuros, seria interessante criar novos experimentos, variando, por exemplo, o número de passeios aleatórios e o limite utilizado para diminuir o tamanho das vizinhanças dos usuários e filmes. Isso permitiria que fosse estudado o impacto desses parâmetros sobre a qualidade das recomendações.

Também seria interessante implementar outros algoritmos de recomendação, não se limitando apenas a algoritmos de filtragem colaborativa, mas incluindo também outras técnicas baseadas em modelo ou conteúdo e probabilísticas. A implementação desses algoritmos permitiria, por exemplo, que outros aspectos do dataset fossem explorados, como o uso dos ids do imdb e do tmdb.

Por fim, o estudo de outras métricas que meçam outras características de uma recomendação também seria proveitoso. Por exemplo, métricas de eficiência dos algoritmos, estabilidade e "novidade"das recomendações.

Referências Bibliográficas

[1] **360pi(2016)** 360pi. How many products does amazon carry?, 2016. URL https://0ca36445185fb449d582-f6ffa6baf5dd4144ff990b4132ba0c4d.ssl.cf1.rackcdn.com/IG_360piAmazon_9.13.16.pdf. Visitado em 2019-11-07. Citado na pág. 1

[2] **Harper e Konstan(2015)** F. Maxwell Harper e Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19. ISSN 2160-6455. doi: 10.1145/2827872. URL http://doi.acm.org/10.1145/2827872. Citado na pág. 17

[3] **Herlocker** *et al.***(2004)** Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen e John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53. ISSN 1046-8188. doi: 10.1145/963770.963772. URL http://doi.acm.org/10.1145/963770.963772. Citado na pág. 4, 14

[4] **Internet Movie Database(1990)** Internet Movie Database. Internet movie database, 1990. URL https://www.imdb.com. Visitado em 2019-11-04. Citado na pág. 17

[5] **Jannach** *et al.***(2010)** Dietmar Jannach, Markus Zanker, Alexander Felfernig e Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, New York, NY, USA, 1st edição. ISBN 0521493366, 9780521493369. Citado na pág. 14

[6] **Neo4J(2019)** Neo4J. Neo4j, 2019. URL https://neo4j.com. Visitado em 2019-11-17. Citado na pág. 18

[7] **Netflix(2009)** Netflix. Netflix prize, 2009. URL https://www.netflixprize.com/index.html. Visitado em 2019-11-03. Citado na pág. 2

[8] **Node.js(2019)** Node.js. Node.js, 2019. URL https://nodejs.org/en/. Visitado em 2019-11-17. Citado na pág. 18

[9] **Sarwar** *et al.***(2001)** Badrul Sarwar, George Karypis, Joseph A Konstan e John Riedl. Item-based collaborative filtering recommendation algorithms. Em *Proceedings of the 10th International Conference on World Wide Web, WWW 2001*, WWW '01, páginas 285–295. Association for Computing Machinery, Inc. ISBN 1581133480. doi: 10.1145/371920.372071. Citado na pág. 30

[10] **Schwartz(2003)** B. Schwartz. *The Paradox of Choice: Why More Is Less*. Harper Perennial. HarperCollins. ISBN 9780060005689. URL https://books.google.com.br/books?id=zutxr7rGc_QC. Citado na pág. 1

[11] **The Movie Database(2008)** The Movie Database. The movie database, 2008. URL https://www.themoviedb.org. Visitado em 2019-11-04. Citado na pág. 17