Capítulo 1

Conceitos, Implementação e Dados Privados de Algoritmos de Recomendação

Leonardo Herdy Marinho, Rodrigo Campos, Rodrigo Pereira dos Santos, Mônica Ferreira da Silva e Jonice Oliveira

Abstract

Through the recommendation algorithms, it is possible to suggest items relevant to users, increasing the proximity to their interest. These facilities also aim to reduce the time that would be spent searching for desired items. These algorithms can be applied in many scenarios, presenting relevant results in solving various real-word problems. In this context, the purpose of this chapter is to simplify and present the concepts of recommendation algorithms, demonstrating how these techniques work. Concepts and challenges involving data privacy in these algorithms are also presented. Finally, this chapter introduces Python programming language operations and applies the recommendation techniques of the collaborative filtering approach, using the cosine similarity.

Resumo

Por meio dos algoritmos de recomendação, é possível sugerir itens relevantes para usuários, aumentando a proximidade com o interesse dos mesmos. Essas facilidades visam também reduzir o tempo que seria dispensado na busca de itens desejados. Esses algoritmos podem ser aplicados em muitos cenários, apresentando resultados relevantes na solução de diversos problemas. Nesse contexto, o objetivo deste capítulo é simplificar e apresentar os conceitos sobre algoritmos de recomendação, demonstrando como essas técnicas funcionam. São apresentados ainda, conceitos e desafios envolvendo a privacidade de dados nesses algoritmos. Por fim, esse capítulo apresenta operações com linguagem de programação Python e aplica as técnicas de recomendação da abordagem filtragem colaborativa, utilizando a similaridade cosine.

1.1. Introdução

Os algoritmos de recomendação implementam filtros de informação visando apresentar itens ou objetos como: páginas web, filmes, músicas, livros, medicamentos, lojas e artigos que provavelmente são do interesse do usuário. Algoritmos de recomendação são amplamente utilizados por grande parte das gigantes redes lojistas, sites focados em entretenimento, redes sociais, players de música e mais uma gama de prestadores de serviços ou vendedores de produtos. É um tema em alta em toda a comunidade científica e mercadológica. O conjunto desses algoritmos e técnicas é chamado de sistema de recomendação.

Os algoritmos de recomendação podem agir sem o acionamento do usuário e por essa razão, alguns usuários podem não notar que determinado *website* ou sistema tenha um algoritmo de recomendação. Nos últimos anos, muitas aplicações que envolvem comércio eletrônico e buscas utilizam algum tipo de mecanismo de recomendação.

Os algoritmos de recomendação têm desempenhado um importante papel na solução do problema de sobrecarga de informações [Ricci et al. 2015]. Entretanto, é preciso considerar também que esses algoritmos apresentam riscos de privacidade e algumas preocupações devem ser observadas na implementação e utilização de dados pelos recomendadores [Feng et al. 2018].

Nesse contexto, esse capítulo explora conceitos dos algoritmos de recomendação, buscando introduzir o assunto e suas principais técnicas. São tratados exemplos do cotidiano que utilizam esses algoritmos, bem como quais são os principais tipos de algoritmos, organizados após uma revisão da literatura. Considerando a utilização da linguagem Python em diversas soluções de recomendação, bem como a variedade de bibliotecas para ciência de dados, esse capítulo introduz as principais operações da linguagem Python. Dessa forma, implementamos um algoritmo de recomendação utilizando métodos de similaridade “cosine”.

Esse capítulo está organizado da seguinte forma: a Seção 1.2 aborda os conceitos dos algoritmos de recomendação, destacando o surgimento e histórico, bem como os principais conceitos do processo de busca e recuperação da informação; a Seção 1.3 apresenta exemplos da aplicação dos algoritmos de recomendação na indústria, bem como os diferenciais a nível de usuário de cada uma das soluções; a Seção 1.4 apresenta os principais tipos de abordagens de recomendação e as diferenças investigadas na literatura; a Seção 1.5 aborda alguns problemas e soluções para atender a privacidade de dados nos recomendadores; a Seção 1.6 tem um enfoque em conhecimentos básicos sobre Python; na Seção 1.7 é apresentado como implementar um recomendador de filmes em Python utilizando um *dataset* pré-selecionado; as principais oportunidades de pesquisa e desafios em sistemas de recomendação são abordados na Seção 1.8; e por fim, a Seção 1.9 conclui o capítulo com considerações finais.

1.2. Algoritmos de Recomendação

Os usuários da internet buscam constantemente por informações que possam auxiliá-los em seu cotidiano (Figura 1.2). Com a grande massa de dados, produtos, serviços e opções que estão disponíveis, encontrar o que realmente é procurado se tornou uma tarefa árdua [Cazella et al. 2008].

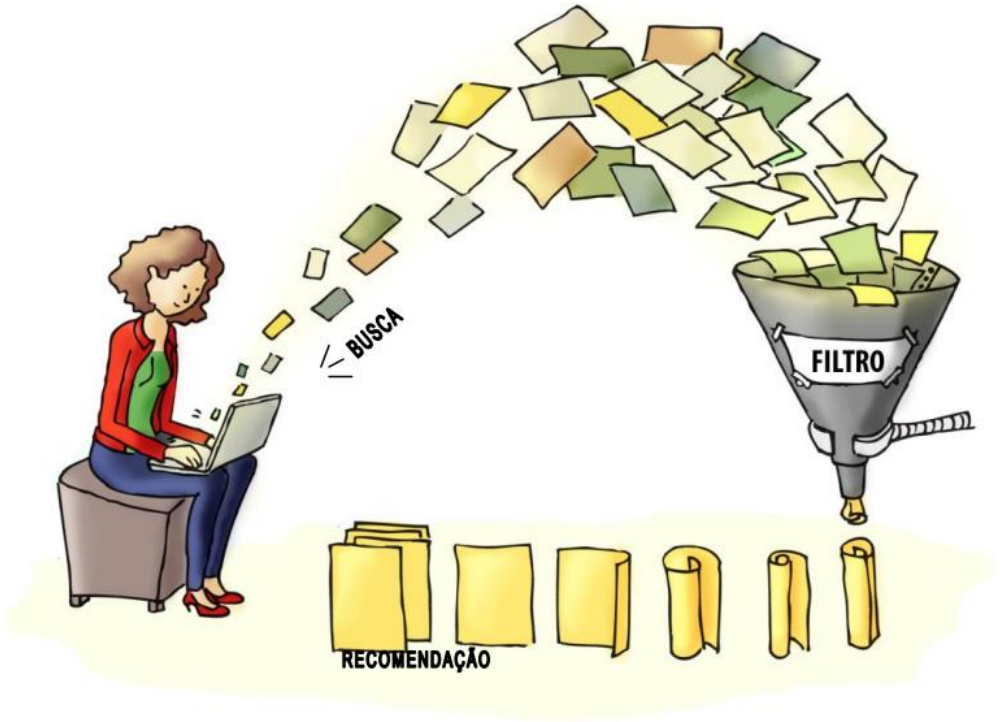

Figura 1.2. Funcionamento de um algoritmo de recomendação

Fonte: [Motta et al. 2011]

Os recomendadores surgem com o objetivo de propor soluções para essa tarefa de recuperar uma informação de acordo com a necessidade do usuário. Essa seção apresenta como esses sistemas evoluíram desde o surgimento, bem como os conceitos do processo de busca de uma determinada informação. Por fim, é apresentado como funciona a lógica de recomendação.

1.2.1. Dados Históricos

A necessidade de construir algoritmos que recomendem algo é bem recente se compararmos com o tempo histórico que a computação existe. O nascimento de tal necessidade teve seu início com o aumento do poder de armazenamento de dados ocorrido principalmente no início da década de 90, que dificultou aos usuários encontrarem de forma rápida e fácil algo que buscam em uma pesquisa realizada na *web*. Anteriormente a análise realizada pelo usuário era bastante simples, afinal, as buscas retornavam um valor substancial relativamente pequeno de itens que eram facilmente triados pelo usuário em poucos minutos. Tal realidade mudou no início dos anos 90 com o advento da Internet e a interconexão entre milhares senão milhões de pessoas, serviços, sensores e mais uma gama de geradores de dados. A cada dia mais dados eram gerados e armazenados, tornando assim, complexa a tarefa de encontrar o que era relevante ao usuário para ser exibido logo nas primeiras páginas de resultados das buscas.

Com o surgimento de serviços de música online, vídeo, venda de produtos e afins, aumentou o desafio de não tomar o tempo dos usuários com conteúdo irrelevante ao gosto deles. As grandes companhias que investiam no “setor online” entenderam que o desperdício do tempo dos usuários ocasionava a redução do consumo online, ou seja,

quanto mais tempo um usuário desperdiçasse para encontrar algo que estava buscando maiores eram as chances de desistência. Isso desencadeou a corrida para o "algoritmo perfeito", mais conhecido como o algoritmo de recomendação, que trouxesse o mais rápido possível exatamente o que o usuário esperava encontrar.

Um dos primeiros sistemas com algoritmos de recomendação conhecido foi o RINGO [Shardanand e Maes 1995]. Segundo Motta et al. (2011), o RINGO trabalhava com a recomendação de músicas a partir do perfil do usuário. As recomendações eram criadas partindo das informações explicitamente fornecidas pelos próprios usuários, ou seja, os usuários precisavam dizer que não gostavam de um ritmo *r* para que o algoritmo não exibisse músicas desse ritmo, por exemplo. O algoritmo ajustava as recomendações de músicas continuamente a partir do uso prolongado, além de realizar sucessivas recomendações analisando o feedback dos usuários. O Grouplens criado aproximadamente na mesma época que o RINGO, na década de 90, implementou uma arquitetura genérica para as recomendações de notícias. Mais tarde, o sistema do Grouplens foi evoluído para o MovieLens [Konstan et al. 1997] que consistia em trabalhar com a sugestão de filmes geradas a partir da correlação entre a avaliação dos usuários sem utilizar características ou parâmetros definidos previamente pelo usuário. Esses trabalhos foram de suma importância para consolidar o marco inicial da implementação e evolução dos atuais algoritmos de recomendação. Ao final desse capítulo, utilizamos um conjunto de dados do MovieLens para explorar recomendações de filmes.

A tecnologia construída para o funcionamento do RINGO posteriormente foi transformada em um produto chamado Firefly, um site de recomendação de músicas, artistas e livros. Após inúmeras parcerias, o Firefly difundiu o sistema de recomendação e a filtragem colaborativa. Posteriormente grandes empresas adotaram a ideia da tecnologia de recomendação do Firefly. como o Yahoo, Amazon e eBay.

1.2.1. Busca e Recuperação da Informação

Recuperação de Informação (RI) é geralmente considerada um sinônimo de recuperação de documentos, mas na verdade, os processos dos sistemas de RI produzem a recuperação do que seria o objetivo principal do usuário. As abordagens desenvolvidas para esse fim são aplicáveis a uma série de tarefas relacionadas ao processamento de informações existentes, tanto na recuperação de dados quanto na recuperação de conhecimento [Jones e Willett 1997]. Dentre essas possibilidades de aplicação, estão os algoritmos de recuperação

A recuperação de dados, no contexto de um sistema de RI, consiste em identificar quais documentos em uma determinada coleção contêm as palavras-chave da consulta do usuário. No entanto, o usuário está interessado em recuperar determinadas informações sobre um assunto, ou seja, dados em um contexto desejado, e não apenas em recuperar dados que satisfaçam uma consulta. Devido a isso, a estratégia de recuperação surge para resolver essa tarefa subjetiva. Essa estratégia consiste no uso de um algoritmo, no processo de recuperação e classificação de sistemas de RI, que identifica o coeficiente de similaridade entre uma consulta de usuário em um determinado contexto e um conjunto de documentos coletados [Santos e Bräscher 2017].

O termo recuperação de informação refere-se a uma pesquisa que pode abranger qualquer forma de informação: dados estruturados, texto, vídeo, imagem, som, notas musicais ou sequências de DNA [Grossman e Frieder 2004]. Em [Baeza-Yates e Ribeiro-Neto 2013], é destacado que a RI fornece aos usuários acesso fácil a informações de interesse, lidando com representação, armazenamento, organização e acesso a itens de informação que devem fornecer aos usuários facilidade de acesso a informações. Portanto, o objetivo principal da RI é recuperar todos os documentos relevantes às necessidades de informações do usuário e, ao mesmo tempo, recuperar o mínimo possível de documentos irrelevantes. Portanto, o principal desafio não é apenas extrair informações dos documentos, mas ter uma noção de sua relevância.

As estratégias da RI para atribuir uma medida de similaridade entre uma consulta e um documento são baseadas no senso comum de quão mais frequente um termo é encontrado em um documento e em uma consulta, mais relevante é o documento para consulta. Uma estratégia de recuperação consiste em um algoritmo que processa uma consulta Q e um conjunto de documentos D1, D2, ..., Dn e identifica o coeficiente de similaridade para cada um dos documentos 1 <= i <= n. As estratégias de recuperação de informações são: Modelo de Vetor Espacial, Recuperação Probabilística, Modelos de Linguagem, Redes de Inferência, Indexação Booleana, Indexação Semântica Latente, Redes Neurais e Algoritmos Genéticos e Recuperação de Conjuntos Fuzzy [Grossman e Frieder 2004].

O sistema de RI consiste em obter a coleção de documentos - prontos ou coletados na web. Depois disso, o sistema de RI rapidamente armazena e indexa a coleção de documentos, para recuperar e classificar. A estrutura de índice mais conhecida é a lista invertida que consiste em todas as palavras da coleção. Cada palavra tem uma lista dos documentos em que está presente. Após o processo de indexação, o processo de recuperação é iniciado, procurando documentos que atendam à consulta do usuário.

Posteriormente, a consulta é analisada sintaticamente e expandida com formas variantes de palavras de consulta. A consulta expandida usa um índice para recuperar um subconjunto dos documentos. Esses documentos recuperados são classificados e os que estão no topo da classificação são retornados ao usuário. O processo de classificação consiste em identificar os documentos com maior probabilidade de serem relevantes para o usuário. E considerando a subjetividade inerente à escolha do critério de classificação, torna-se necessário um processo de avaliação para analisar a qualidade dos resultados produzidos pelo sistema de RI.

1.2.2. Lógica de Recomendação

Algoritmos de recomendação analisam em um espaço muito vasto e denso de dados o que os usuários desejam, para isso os algoritmos levam em consideração a análise crítica com base no perfil dos usuários, padrões comportamentais, itens visualizados e mais uma série de outros fatores que fazem o algoritmo entender quais resultados da busca são em teoria mais relevantes para quem está realizando a mesma. Os resultados muitas vezes não são completamente precisos, mas trazem dentre os primeiros itens da lista o que provavelmente o usuário deseja. A cada ano tais algoritmos são aperfeiçoados de modo que a acurácia se mantém cada vez mais elevada. Na Figura 1.1 temos um exemplo simples de como um sistema de recomendação funciona na prática.

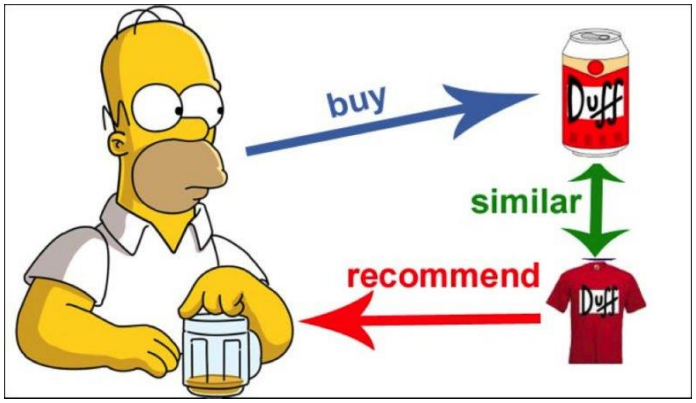

Figura 1.1. Exemplo do funcionamento da recomendação

Após o usuário, que na Figura 1.1 é representado pelo personagem de desenho animado Homer Simpson, comprar uma cerveja da marca Duff o algoritmo de recomendação compara com outros produtos que dispõe em sua base de dados e percebe que existe um outro produto que pode ser interessante ao usuário, logo, este define uma similaridade entre a bebida e a camisa Duff. Ao ter um alto valor de similaridade entre dois itens de acordo com o que o algoritmo estima que o usuário terá interesse, ele recomenda a camisa com a marca da cerveja. Nesse contexto, as recomendações buscam sempre o resultado mais assertivo, mas um algoritmo não necessariamente conseguirá prever todos os casos possíveis. Se Homer tivesse comprado uma camisa Duff naquele dia, o algoritmo teria mostrado uma recomendação muito boa, porém o personagem não compraria a mesma tendo em vista que já tem uma daquele modelo. O algoritmo acertou no gosto do usuário, mas errou na hora da exibição já que não tinha em sua base de dados como prever que o usuário já havia obtido aquele produto anteriormente. Assim funcionam diversos tipos de algoritmos de recomendação. Há sempre a possibilidade de encontrarem dificuldades como essa no processo. No geral, tais algoritmos são bastante satisfatórios já que poupam horas de pesquisas e navegação em páginas, mas ainda existem desafios em abertos para contemplarem todos os cenários possíveis.

Esses desafios envolvem uma dificuldade de se ter acesso às informações que apoiam decisões corretas em um grande conjunto de dados, como os de empresas de vendas. As técnicas de RI auxiliam nesta árdua tarefa. O primeiro aspecto a ser observado é identificar o que um usuário está buscando. Basicamente um usuário menciona um grupo de palavras e/ou símbolos. Em seguida, os documentos/páginas/registros identificados como os mais relevantes são recomendados por algoritmos de busca e recuperação. O princípio dos sistemas de recomendação é baseado em relevância, logo, "o que é relevante para mim também pode ser relevante para alguém com interesses e gostos similares". A grande maioria das técnicas de recomendação trabalham com as avaliações realizadas pelos indivíduos, assim, é possível identificar usuários que avaliaram itens (com notas, por exemplo) de forma similar, passando a entender, portanto, os gostos coletivos. Nesse contexto, o algoritmo consegue criar médias para as pessoas que tem gosto similares (Figura 1.3). Quando um novo indivíduo que se assemelha a um determinado grupo entra em cena, o algoritmo o sugere algo que os outros pertencentes do grupo demonstram interesse. Assim, as chances de acerto são potencializadas partindo da premissa que os gostos do grupo são próximos.

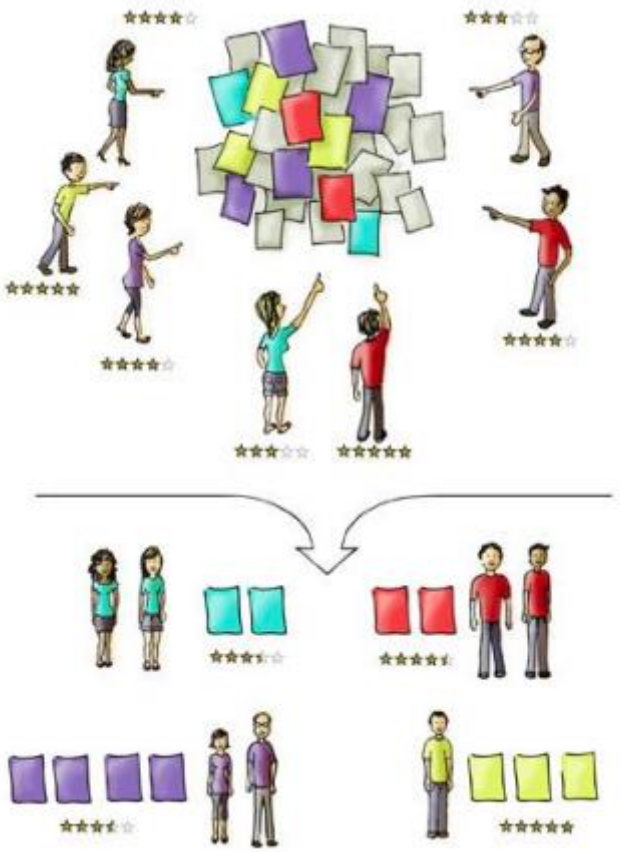

Figura 1.3. Média por grupos similares

Fonte: [Motta et al. 2011]

1.3. Exemplos de Uso

Sistemas de recomendação são amplamente utilizados por grande parte dos sites que nos rodeiam. Quando lemos notícias online, de alguma forma os sistemas de recomendação estão presentes. Quando acessamos as redes sociais esses sistemas podem existir sugerindo publicações que julgam mais relevantes para os usuários. O mesmo pode ocorrer em lojas virtuais, com a presença de sistemas de recomendação exibindo produtos que estão no raio de interesse maior dos clientes. Nesta seção mostraremos dois exemplos de algoritmos de recomendação aplicados.

1.3.1. Portal de Notícias G1

Sites de notícias são excelentes exemplos da aplicação de recomendação. Desde o início da democratização da internet as notícias são disseminadas em sites ou veículos similares. Exibir aos leitores conteúdos que são relevantes, tornam maior a permanência dos mesmos na página. Nesse sentido, a fidelização e as receitas aumentam, tendo em vista que quanto mais tempo um usuário navegar pelo site mais anúncios intercalados com as matérias normalmente serão exibidos. Nada impede de que junto às notícias sejam exibidas recomendações de outras notícias, eventos ou qualquer conteúdo que o

algoritmo da página julgar interessante para o leitor. Um grande veículo de comunicação jornalística no Brasil é o portal de notícias G1[1].

No caso do G1, é possível encontrar recomendações ao navegar por uma notícia, por exemplo. Ao final de cada página da notícia, existe uma recomendação para que o leitor siga lendo uma outra notícia que o algoritmo de recomendação presente no site entende como relevante para aquele leitor naquele momento. Temos tal exemplo demonstrado na Figura 1.4.

Figura 1.4. Exemplo de recomendação de notícias

Fonte: Captura de tela realizada pelas pessoas autoras no dia 31 de outubro de 2019.

1.3.2. Netflix

Seguindo com os exemplos, também é de suma importância citar a Netflix como uma plataforma que faz uso de sistemas de recomendação.A Figura 1.5 mostra que na tela principal de usuários da Netflix há uma seção explícita de recomendação chamada "Sugestões para você", com diversas séries exibidas para um usuário x.

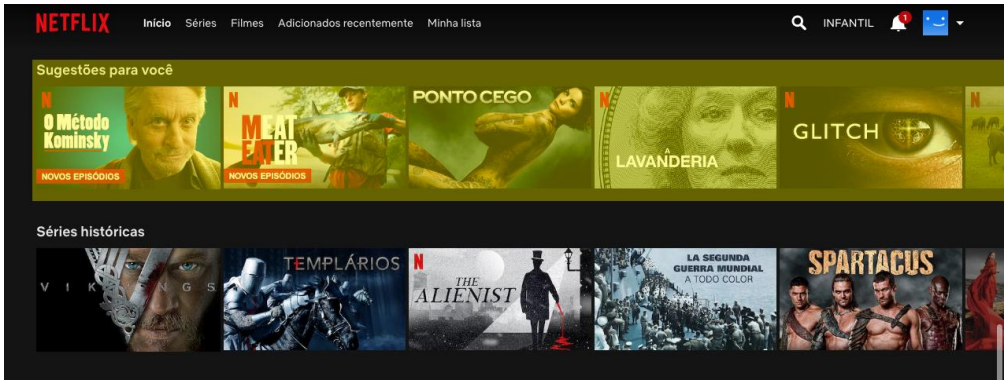

Figura 1.5. Sugestões personalizadas para usuário x da Netflix

Fonte: Captura de tela realizada pelas pessoas autoras no dia 31 de outubro de 2019.

[1] https://g1.globo.com

Atualmente, a Netflix é uma das maiores plataformas para quem deseja assistir filmes, séries, documentários e similares. Seu enorme sucesso se deu pela habilidade de manter seus usuários engajados com o conteúdo que oferecem. Isso se deu também com a alta precisão do algoritmo de recomendação que construíram, que permite sugerir o que um usuário talvez goste para assistir posteriormente. É possível notar na Figura 1.6 que quando mudamos para um usuário y na plataforma, automaticamente outras séries completamente diferentes são inseridas na lista de sugestões visando oferecer ao usuário um conteúdo mais próximo de seu perfil. Portanto, essas séries são completamente distintas das séries mostradas na Figura 1.5 para o usuário x.

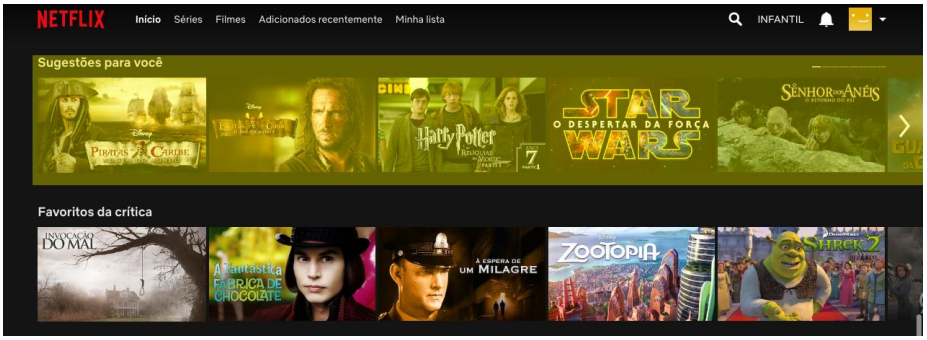

Figura 1.6. Sugestões personalizadas para usuário y da Netflix

Fonte: Captura de tela realizada pelas pessoas autoras no dia 31 de outubro de 2019.

É interessante destacar que tais algoritmos utilizam de várias estratégias para definir qual filme ou série devem recomendar. No caso da Netflix, utilizam-se dos dados de navegação do usuário como: palavras inseridas na busca, filmes clicados, filmes assistidos por completo anteriormente, filmes "abandonados" pela metade, conteúdos curtidos pelo usuário, conteúdos adicionados na lista de preferências e mais uma série de dados que coletam ao longo da permanência do usuário na plataforma.

A Netflix é uma das plataformas que exibe ao usuário o coeficiente de relevância empregado pelo seu algoritmo. Sempre que abrimos uma série ou filme, por exemplo, a plataforma exibe um índice de relevância deste item para o usuário. Como exemplo, a Figura 1.7 exibe o resumo do filme "Piratas do Caribe no Fim do Mundo". Além da descrição, a plataforma mostra para o usuário x que esse filme é 82% relevante de acordo com os dados por eles analisados.

Figura 1.7. Relevância de 82% para o usuário x da Netflix

Fonte: Captura de tela realizada pelas pessoas autoras no dia 31 de outubro de 2019.

Já para o usuário y (Figura 1.8), a plataforma não mensura a relevância tendo em vista que aquele filme, em teoria, não seria do gosto do usuário de acordo com seu perfil.

Figura 1.8. Relevância não mensurada para o usuário y da Netflix

Fonte: Captura de tela realizada pelas pessoas autoras no dia 31 de outubro de 2019.

Com isso, tem-se que a maior parte do catálogo sugerido e resultados de busca utilizam técnicas de recomendação por similaridade de conteúdo. Ao pesquisar por "Piratas do Caribe", são retornados abaixo dos resultados esperados uma série de outros filmes que não são necessariamente ligados ao tema do filme, mas que trazem diversas similaridades como: atores, conteúdo da sinopse, filmes que pessoas que pesquisaram por "Piratas do Caribe" assistiram após recomendação do algoritmo, e mais uma série de fatores. A Figura 1.9 mostra as recomendações que vieram após a busca do usuário x por "Piratas do Caribe" no fim do mundo.

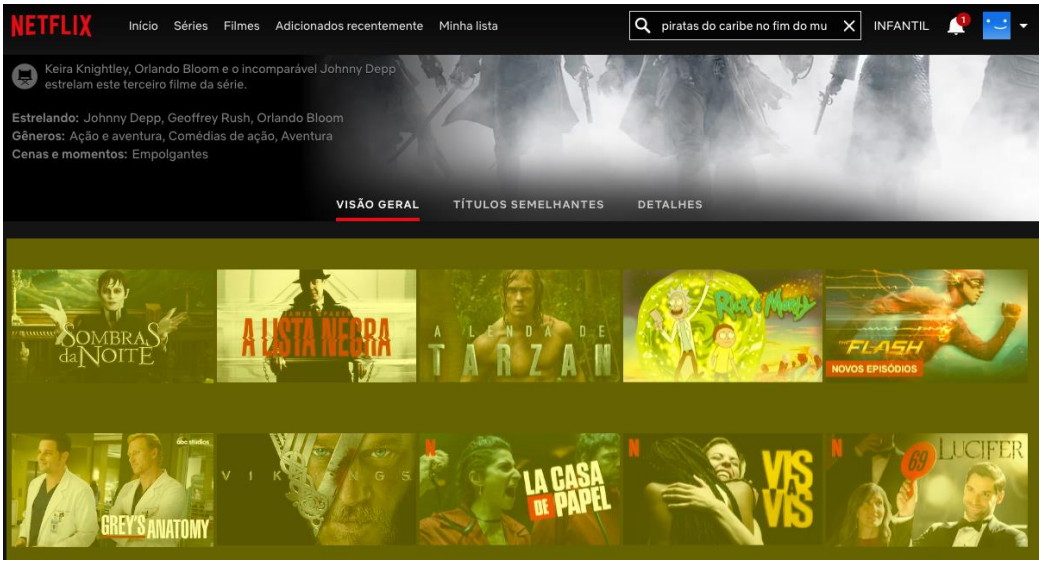

Figura 1.9. Sugestões após a busca por "Piratas do Caribe no Fim do Mundo" na Netflix

Fonte: Captura de tela realizada pelas pessoas autoras no dia 31 de outubro de 2019.

1.4. Tipos de Algoritmos de Recomendação

Segundo Herlocker et al. (2000), por muitos anos os cientistas têm direcionado esforços para buscar soluções e meios de amenizar o problema ocasionado com a sobrecarga de informações geradas através por soluções que reconhecem e categorizam informações automaticamente. Recomendações não são triviais para serem realizadas. Tanto no mundo físico quando no mundo digital é preciso observar uma série de fatores que influenciam um tipo de recomendação. Oferecer comida de cachorro para as pessoas degustarem em um supermercado não é algo interessante, na verdade, seria um grande fracasso. Mas oferecer café por exemplo pode ser uma ótima opção. É possível perceber o quão falha ou bem-sucedida é essa recomendação de "O que oferecer para degustação humana", mas para as máquinas não é. Café e ração são produtos como quaisquer outros. É preciso ensinar para a máquina o sentido das coisas para que ela não sugira um pneu para degustação, por exemplo. Ou, que ela não sugira um desenho animado infantil para uma pessoa que prefira filmes de terror.

Ao longo dos anos foram desenvolvidas técnicas por pesquisadores que refinam os resultados das buscas utilizando diversos tipos de filtros e parâmetros. Tais filtragens são realizadas de acordo com as características de um item, assim como o comportamento dos usuários/consumidores perante aquele item. Logo começaram a entender padrões que poderiam ser utilizados para melhorar as recomendações e torná-las realmente úteis e de alta relevância para quem a recebesse. Para isso, criaram-se técnicas para buscar e recuperar informações, filtrar itens com base nas descrições dos conteúdos e também analisar o comportamento dos usuários perante o item. Assim, foi possível construir um modelo colaborativo para entender melhor como é a interação dos usuários com os itens e gerar modelos estatísticos baseados em grupos similares.

Nos sistemas de recomendação são utilizadas em geral uma das três técnicas de filtragem de informação citadas a seguir: filtragem baseada em conteúdo, filtragem colaborativa, também conhecida como filtragem social e filtragem híbrida [Cazella, Sílvio César et al. 2010].

1.4.1. Filtragem Baseada em Conteúdo

Alguns softwares têm como objetivo gerar de forma automática descrições dos conteúdos dos itens e comparar estas descrições com os interesses dos usuários, verificando assim se o item é ou não relevante [Balabanović e Shoham 1997].

A filtragem baseada em conteúdo (Figura 1.10) consiste em realizar análises comparativas envolvendo o conteúdo de itens distintos, como por exemplo, palavras similares, temas similares, gêneros similares e afins. A análise de diversos componentes do conteúdo de um item torna possível identificar a similaridade entre dois ou mais itens através de um coeficiente numérico. A recomendação baseada na filtragem por conteúdo utiliza informações anteriores do usuário em relação a um item para recomendar itens similares. São recomendados, por exemplo, itens que mais se aproximam aos outros itens avaliados de forma positiva por determinado usuário.

A indexação da frequência de termos, ou seja, o quanto determinados termos se repetem no conteúdo de um item é bastante utilizada nessa abordagem, sendo as informações dos documentos e necessidades dos usuários descritas por vetores que armazenam a frequência com que as palavras ocorrem em um dado documento (como

em [Campos et al. 2019]) ou em uma consulta realizada pelo usuário [Cazella, Sílvio César et al. 2010].

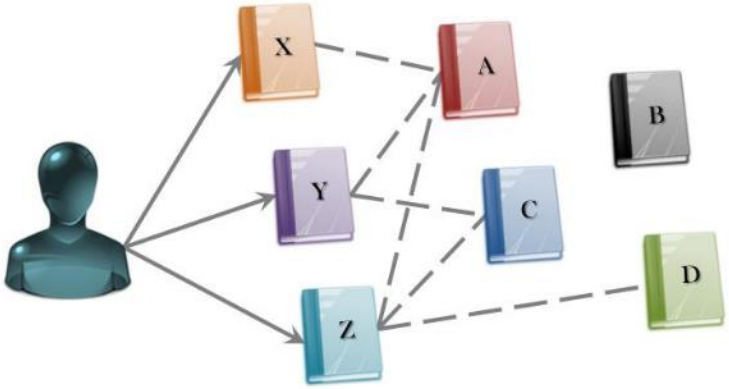

Figura 1.10. Representação da abordagem de filtragem por conteúdo

Fonte: [Costa et al. 2013]

1.4.2. Filtragem Colaborativa

A diferença da abordagem entre a filtragem colaborativa para a baseada em conteúdo está no fato de que, enquanto a filtragem colaborativa considera as preferências de vários usuários analisadas em seu perfil, a baseada em conteúdo considera padrões semelhantes relacionados às experiências apenas do usuário interessado [Aggarwal 2016]. Dado o item i e usuário u, a abordagem de filtragem colaborativa tem como princípio o fato de que, se um usuário u avalia itens semelhantes a outro usuário u', há uma grande chance de a próxima avaliação de um usuário u para um novo item seja semelhante ao do usuário u'. Portanto, se outros usuários avaliaram dois itens de maneira semelhante, existe uma tendência do usuário u avaliá-los da mesma maneira. A abordagem de filtragem colaborativa pode ser dividida em duas etapas; a abordagem em vizinhança e a baseada em modelos [Desrosiers e Karypis 2011]. O primeiro pode ser realizado de duas maneiras:

- Baseado em conteúdo: considere o exemplo com os usuários u e u'. A semelhança entre esses usuários os torna vizinhos. Sempre que o sistema avalia o interesse de u pelo item i, ele verifica a classificação dada pelos vizinhos [Desrosiers e Karypis 2011];
- Baseado em itens: nessa abordagem, o sistema consulta outros itens semelhantes ao item i. Dados esses itens semelhantes, toda vez que o sistema avalia o interesse de u pelo item i, ele considera as avaliações feitas pelo usuário u para esses itens semelhantes. Para definir se dois itens são semelhantes, o sistema verifica se muitos usuários classificaram esses itens da mesma maneira [Desrosiers e Karypis 2011]. É também chamado de abordagem item a item [Koren 2008].

Embora as possibilidades da abordagem de vizinhança sejam baseadas em item ou usuário, o modelo propõe mesclar itens e usuários no mesmo espaço de fatores latentes, permitindo inferir automaticamente com base no feedback do usuário [Koren 2008]. Apesar de ter várias técnicas nessa abordagem [Desrosiers e Karypis 2011], existem algumas muito comuns, como *clustering*, classificação, modelo latente, Markov Decision Process (MDP) e Matrix Factorization (MF) [Do et al. 2010].

A técnica para descobrir de forma automática as relações entre um dado usuário e seus "vizinhos mais próximos" consiste em (1) calcular a similaridade do usuário alvo em relação aos outros usuários; (2) selecionar um grupo de usuários com maiores similaridades para considerar na predição; e (3) normalizar as avaliações computando as predições, ponderando as avaliações dos usuários mais similares [Cazella, Silvio César et al. 2010]. Seguindo a lógica do exemplo de recomendação representado pela Figura 1.10, a Figura 1.11 mostra de forma representativa como é uma recomendação no modelo de filtragem colaborativa.

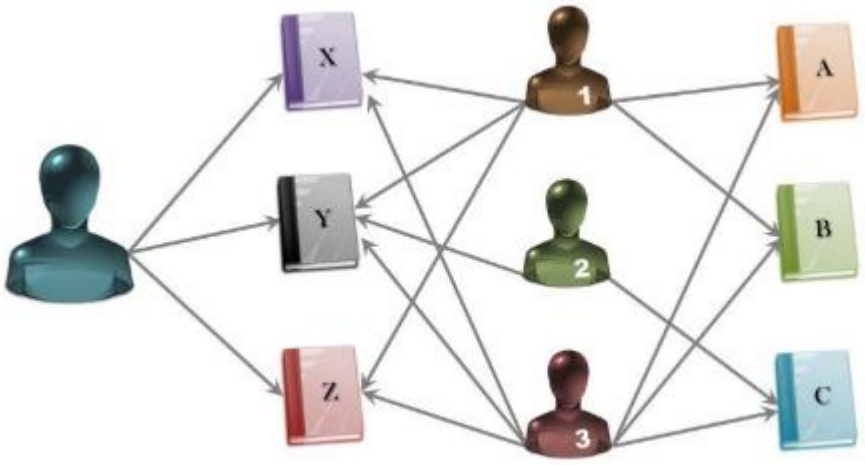

Figura 1.11. Representação da abordagem de filtragem colaborativa

Fonte: [COSTA et al, 2013]

No exemplo da Figura 1.11, um dado usuário alvo gostou dos livros X, Y e Z. É possível notar que seus vizinhos mais próximos são os usuários 1, 2 e 3, tendo em vista que também leram os mesmos livros X, Y, e Z, logo, provavelmente esses usuários têm gostos similares. Portanto, o sistema pode recomendar os livros A, B e C para o usuário alvo, tendo em vista que estes livros foram consumidos pelos vizinhos do usuário alvo.

1.5. Dados Privados em Sistemas de Recomendação

Com o gigante volume de dados armazenados sobre as preferências, itens visualizados, itens comprados, itens ignorados e mais uma gama de dados sensíveis que os sistemas armazenam para refinar as recomendações dos usuários, existe uma preocupação vinculada à segurança. Isso inclui questionamentos sobre como esses dados são tratados e utilizados por estes sistemas. Dados dos quais se armazenados ou utilizados de forma não correta podem expor centenas de milhares de usuários.

A privacidade de dados em algoritmos de recomendação pode incluir o princípio de Privacidade Diferencial. A partir desse recurso matemático, é possível atribuir aos algoritmos de recomendação um valor do quanto esse algoritmo preserva a privacidade dos usuários. Uma dificuldade existente para a obtenção de privacidade é que a privacidade diferencial garante a privacidade de usuários, mas pode diminuir a utilidade do algoritmo de recomendação [Jiang et al. 2019]. Essa teoria de privacidade diferencial começou a ser implementada em sistemas de recomendação de filtragem colaborativa muito recentemente com a proposta de Mcsherry e Mironov (2009), fazendo parte da categoria de "*data transformation technology*", segundo Feng et al. (2018).

Outra categoria descrita em [Feng et al. 2018] para privacidade de dados em sistemas de recomendação é a de recomendação distribuída [Qi et al. 2017]. As soluções dessa categoria podem envolver processos para ocultar informações reais de usuários, ou aplicar dimensões de qualidade através da "*List Scheduling Heuristic*" (LSH). Existe também a possibilidade do uso de encriptação homomórfica. Armknecht e Strufe (2011) sugerem utilizar essa técnica para atender as necessidades de construir um recomendador que não necessite de informações de preferências de usuários, cujo provedor de recomendação não precise revelar informações internas, mas que seja possível retornar uma recomendação correta para o usuário.

Os aspectos de privacidade existem até mesmo em cenários que envolvam dados abertos, exigindo uma atenção por parte dos sistemas de recomendação. Um exemplo disso pode ser observado nos Ecossistemas MOOCs [Campos et al. 2018] (entende-se que MOOC, por definição traduzida, significa "Cursos Online Abertos e Massivo"). Nesse cenário, apesar dos dados de cursos dos provedores e dados dos alunos matriculados serem abertos, há restrições que exigem uma camada de autorização por parte dos recomendadores interessados em extrair esses dados. Essa camada deve permitir que a recomendação envolvendo dados pessoais de um determinado aluno nos MOOCs apenas seja processada caso esse aluno se autentique e autorize a extração dos seus dados.

Apesar das soluções existentes, há uma necessidade eminente de novas abordagens que considerem os aspectos de dados privados, principalmente em sistemas de recomendação baseados em conteúdo. Essa abordagem utiliza apenas dados do usuário interessado em receber a recomendação. Na existência de restrições de acesso a esses dados, a recomendação se tornaria ainda mais limitada, podendo comprometer inclusive o resultado final dos itens retornados. A solução para esse problema pode partir, por exemplo, da integração dos dados do usuário interessado com outras bases de dados (como com dados das redes sociais) e, em seguida, inferência de um perfil do usuário. Essas estratégias permitem, inclusive, a assertividade de um algoritmo de recomendação no caso do *cold-start*, ou seja, usuários ou itens que são novos nas plataformas, não possuindo dados suficientes para uma primeira recomendação.

Um enfoque que os sistemas de recomendação baseados em conteúdo podem ter é na recomendação de publicidade direcionada ou na entrega de cupom direcionada. Para esses casos, Wang et al. (2018) analisam trabalhos que buscam criar mecanismos de proteção e privacidade para usuários. Através desse levantamento é possível identificar, inclusive, a importância da proteção de dados para quem fornece a recomendação. Wang et al. (2018) mencionam que no caso de entrega de cupom direcionada, os cupons não podem ser utilizados por outros usuários além do usuário alvo da recomendação. Acrescentam ainda que, além das técnicas de criptografia, mencionada anteriormente, outras soluções possíveis para uma camada de privacidade nessa abordagem de recomendação seria anonimização, teoria de games, ofuscação ou segmentação local.

1.6. Operações com Python

Essa seção inclui elementos técnicos para compreender algumas operações básicas com a linguagem Python e como elas são utilizadas para lidar com dados, cálculos e exibição de resultados. Esses conhecimentos são fundamentais para a compreensão do algoritmo

de recomendação sugerido na Seção 1.7. Portanto, são introduzidos conceitos como: comentários, exibição de valores, variáveis e tipos de dados, operações aritméticas, operações lógicas, controle de fluxo e funções. Vale ressaltar que não são abordadas informações sobre a instalação local do Python ou configuração de ambiente de desenvolvimento tendo em vista que todo o desenvolvimento e exemplos aplicados neste capítulo são realizados na plataforma do Google Colab[2]. Essa plataforma dispõe de um ambiente previamente configurado e pronto para qualquer usuário. Todos os exemplos empregados nessa seção foram testados e desenvolvidos com o Python na versão 3.6, por ser uma das versões mais recentes e estáveis disponíveis no momento em que este capítulo foi desenvolvido.

1.6.1. Comentários

A linguagem Python assim como a grande maioria das linguagens de programação dispõe de comentários para auxiliar na organização do código. Ao comentar uma linha ela é ignorada pelo interpretador da linguagem quando o código for executado, assim, servindo apenas para programadores lerem e entenderem o que foi comunicado naquele código. Neste exemplo são abordadas as maneiras de se utilizar o comentário em linha, por ser o mais utilizado e o que é empregado em nossos exemplos. Vale ressaltar que também existem comentários de blocos, ou seja, é possível comentar várias linhas de uma única vez.

O comentário de linha serve para criar apenas uma linha comentada. No Python os comentários de linha são criados utilizando o caractere “#”. A Figura 1.12 exemplifica a criação de um comentário na primeira linha de um arquivo no Google Colab.

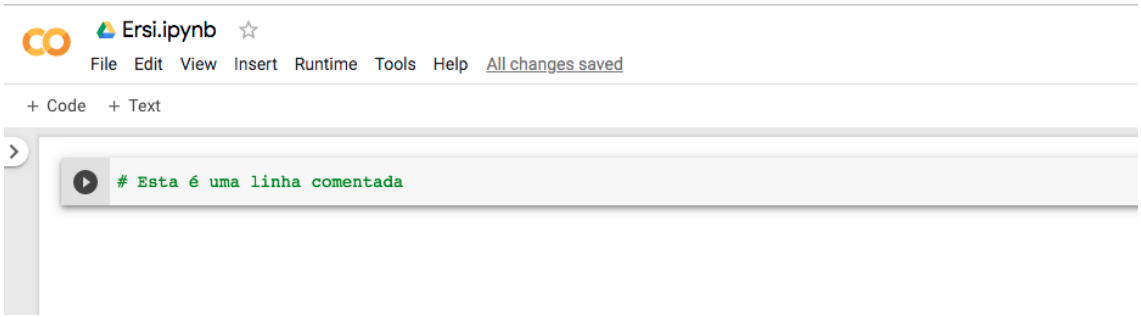

Figura 1.12. Exemplo de comentário de linha no Google Colab

Fonte: Captura de tela realizada pelas pessoas autoras no dia 03 de novembro de 2019.

Normalmente os comentários são utilizados para descrever blocos ou linhas de códigos visando tornar simples a compreensão e a manutenção daquele bloco ou linha por outros programadores. Comentários são amplamente utilizados para documentar funções e cabeçalhos de arquivos. Nos blocos de código das próximas seções, os comentários são utilizados para detalhar passo a passo o que faz cada linha de código escrita, visando auxiliar no entendimento do conteúdo aplicado.

[2] https://colab.research.google.com/

1.6.2. Exibir Textos e Valores

O Python, assim como outras linguagens, dispõe de um comando em especial que serve para exibir valores na tela para que um usuário do sistema ou um programador consiga imprimir valores para o entendimento do que está ocorrendo em determinado momento da execução do algoritmo. Esse recurso é possível através do comando “print”. A Figura 1.13 demostra como utilizar o comando “print” para exibir o texto “Hello World” (que no Python é tratado como um tipo *String*).

```
print("Hello World")
```

Figura 1.13. Código Python para exibição de texto

O comando “print” não se limita à exibição de textos. Também é possível exibir outros tipos de valores, como é abordado nas próximas sessões.

1.6.3. Variáveis e Tipos de Dados

Variáveis são amplamente utilizadas na programação para armazenar valores. Pode-se entendê-las como “caixinhas” que guardam valores na memória do computador. As variáveis no Python têm tipagem dinâmica, ou seja, não é preciso especificar o tipo do valor que estamos atribuindo para ela no ato de declarar a variável. Basicamente, criamos uma variável de tipo genérico e é possível atribuir a ela um texto ou valor numérico, por exemplo. Em linguagem com a tipagem não dinâmica é preciso declarar que determinada variável recebe apenas texto, por exemplo, não sendo possível inserir valores diferentes dos textuais. A Figura 1.14 demonstra como é a atribuição de valores de diferentes tipos a uma variável chamada “a”.

```
a = 1 # Número inteiro
a = 2.65 # Número decimal
a = 1E3 # Notação científica, equivale a 10^3 = 1000.0
a = "Escola regional de sistemas 2019" # Texto (string atribuída com aspas duplas)
a = 'ERSI2019' # Texto (string atribuída com aspas simples)
a = True # Tipo lógico [booleano] (verdadeiro -> True, falso -> False)
a = [1, 2, 3] # Lista de valores numéricos
a = {"a": 1, "b": 2, "c": {"x": 10}} # Dicionário de dados (pares de chave e valor)
```

Figura 1.14. Código Python para exemplificar declaração e exibição de variáveis

No Python o próprio interpretador testa e verifica o tipo da variável para tratá-la da melhor e mais performática forma possível. Na criação de uma variável, é possível declará-la com valores inteiros, decimais, textuais, listas, dicionários e afins e isso torna bem mais simples o uso de variáveis se comparado com as linguagens que exigem tipos fixos. Utilizando as variáveis, é possível realizar as mais diversas operações matemáticas, assim como receber objetos maiores como listas e valores provenientes de arquivos ou banco de dados.

1.6.4. Operações Aritméticas

As operações aritméticas são importantes para diversas linguagens de programação. Com tais operações, é possível calcular praticamente todo tipo de conta matemática básica. Pode-se entender como operadores básicos aritméticos os operadores de soma, subtração, multiplicação e divisão. Pode-se também estender os operadores aritméticos para os operadores um pouco mais avançados como os de exponenciação e cálculo de parte inteira. A Figura 1.15 exemplifica a realização das quatro operações básicas (soma, subtração, multiplicação e divisão) em Python e também como atribuir valores das operações em variáveis para posteriormente exibi-los através do comando "print". No exemplo da Figura 1.15 também é demonstrado como o comando "format" pode auxiliar na apresentação de resultados de uma forma mais agradável e simples de entender visualmente, mesclando texto e valores numéricos.

```
a = 5
b = 2
soma = a + b
# O método format serve para criar uma string que contem campos entre chaves que são substituídos pelos valores que estão dentro do método. No exemplo abaixo, o ".format()" recebe a variável soma
print("Soma: {}".format(soma))
subtracao = a - b
print("Subtração: {}".format(subtracao))
multiplicacao = a * b
print("Multiplicação: {}".format(multiplicacao))
divisao = a / b
print("Divisão: {}".format(divisao))
```

Figura 1.15. Código Python para exemplificar operações aritméticas básicas

Como mencionado anteriormente, os operadores aritméticos não se resumem aos básicos das quatro operações. Também existem mais alguns, dentre os quais ressaltamos os de exponenciação e de parte inteira. A Figura 1.16 ilustra como utilizar tais operadores.

```
a = 5
b = 2
exponenciacao = a**b
print("Valor exponencial: {}".format(exponenciacao))
parte_inteira = a // b
print("Parte inteira: {}".format(parte_inteira))
```

Figura 1.16. Código Python para exemplificar operações aritméticas avançadas

1.6.5. Operações de Comparação

As operações de comparação retornam resultados booleanos, ou seja, *True* (para verdadeiro) ou *False* (para falso). Os comparadores permitem testar se um valor x é igual a um valor y, por exemplo. A Figura 1.17 ilustra como utilizar os operadores de comparação do Python.

```
a = 10
b = 5

# Operador de maior que >
maior_que = a > b
print("{} é maior que {}? {}".format(a, b, maior_que))

# Operador de menor que <
menor_que = a < b
print("{} é menor que {}? {}".format(a, b, menor_que))

# Operador de maior ou igual >=
maior_igual = a >= b
print("{} é maior ou igual a {}? {}".format(a, b, maior_igual))

# Operador de menor ou igual >=
menor_igual = a <= b
print("{} é menor ou igual a {}? {}".format(a, b, menor_igual))

# Operador de igualdade ==
igual_a = a == b
print("{} é igual a {}? {}".format(a,b, igual_a))

# Operador de diferença !=
diferente_de = a != b
print("{} é diferente de {}? {}".format(a,b, diferente_de))
```

Figura 1.17. Código Python para exemplificar operações de comparação

É possível observar na Figura 1.17 que além de verificar igualdade entre dois valores, os comparadores permitem ainda testar se são “diferentes”, “maior que”, “menor que”, “maior ou igual a” e “menor ou igual a”. Com os operadores de

comparação é possível saber se a nota dada por um determinado usuário a um item é maior que a nota que esse mesmo usuário deu para um outro item, ou ainda se a nota de ambos é igual. Diversas combinações podem ser realizadas por sistemas de recomendação para analisar se dado item é mais ou menos similar a outros para que seja criada uma recomendação. Assim, os operadores de comparação cooperam ativamente para os sistemas de recomendação.

1.6.6. Operações Lógicas

Os operadores lógicos unem expressões lógicas formando assim, uma nova expressão que é composta por duas ou mais subexpressões. O resultado lógico (verdadeiro ou falso) de expressões compostas é a relação entre as subexpressões resultando em uma única resposta. A Figura 1.18 demonstra as diferenças no uso dos operadores “and” e “or”.

```
# Operadores and e or


print("Operador and")
print(True and False)
print(True and True)
print(False and True)
print(False and False)


print("\n\nOperador or")
print(True or False)
print(True or True)
print(False or True)
print(False or False)
```

Figura 1.18. Código Python para exemplificar operações lógicas

O Python implementa os valores lógicos utilizando a lógica *booleana*, implementando os valores *True* ou *False*. Ambos precisam necessariamente ser informados com a primeira letra maiúscula. Ao analisar a Figura 1.18 é possível perceber que sempre que existie pelo menos um valor *False*, o operador “and” logo retorna que a expressão é falsa. Já o operador “or” verifica se pelo menos um dos dois valores é positivo. Caso seja, o Python interpreta que a expressão pode ser considerada verdadeira.

Os valores lógicos são de fato a base de toda a computação, até porque, temos que, o bit está ligado ou o bit está desligado. O valor lógico em sua forma mais primitiva assume o número 1 quando for um valor verdadeiro e o valor 0 quando for um valor falso. Toda expressão avaliada na computação de maneira geral, resulta em um valor lógico, isto é, ou a expressão é verdadeira ou falsa.

Além disso, é possível combinar expressões comparativas utilizando operadores booleanos, assim, buscando resultados mais complexos que posteriormente podem ser utilizados em condicionais, por exemplo. A Figura 1.19 demonstra como combinar comparadores de igualdade com os operadores “and” e “or” buscando um resultado único booleano por expressão.

```
# Operações avançadas com operadores lógicos
a = 1
b = 2
c = 3
d = 3
ab = a == b
cd = c == d
print(ab and cd)
print(ab or cd)
```

Figura 1.19. Código Python para exemplificar Exemplo de operações relacionais e lógica combinada

1.6.7. Controle de Fluxo

O controle de fluxo é uma parte importantíssima das linguagens de programação, uma vez que dão total autonomia ao desenvolvedor para definir se algo deve ou não ser realizado. O controle de fluxo só é possível devido às operações lógicas que tornam possível realizar testes de verdadeiro ou falso para dadas expressões. Os conceitos de controle de fluxo podem ser divididos em duas partes: o controle de fluxo através de condicionais e através de laços de repetição. Sobre o controle de fluxo através de condicionais, a Figura 1.20 demonstra como desenvolver um código para tomar certas decisões dependendo de condições prévias.

```
# Utilização de condicionais (if, else, elif)
a = 2
b = False


if b == True:
  print("Eba, o B é positivo!")
elif a > 0:
  print("Eba, o A é maior do que 0")
else:
  print("Poxa... Nenhuma das condições foi verdadeira")
```

Figura 1.20. Código Python para exemplificar o uso de condicionais

Uma das capacidades de uma linguagem de programação é a de decidir o fluxo por onde os comandos são executados. O controle condicional é necessário em praticamente todo programa de computador, sendo basicamente a capacidade de tomar decisões com base em certas condições.

A linguagem Python tem algumas palavras reservadas que são dedicadas especialmente para criar fluxos de execuções condicionais, como: “if”, “else” e “elif”. Basicamente, os condicionais nos permitem selecionar certos blocos de código que são ou não executados dependendo da condição inserida para ser analisada.

Na Figura 1.20, o comando “if” verifica se a primeira condição é válida, ou seja, se a variável “b” possui o valor *True*. Como “b” tem o valor False, a condição foi considerada falsa. O interpretador então pula para o próximo comando, sendo esse o “elif”. A condição do “elif” é verdadeira, logo, o interpretador exibe a mensagem "Eba, o A é maior do que 0". Caso a variável “a” não fosse maior do que 0, o interpretador Python pularia para a próxima instrução que seria o “else”, e consequentemente exibiria o texto "Poxa... Nenhuma das condições foi verdadeira". O comando “else” sempre é executado quando todas as condições acima não forem verdadeiras. O comando “elif” do Python tem a mesma função do comando “elseif” em outras linguagens como o PHP e JavaScript, por exemplo.

Já o controle de fluxo através dos laços de repetição, é a capacidade de repetição de comandos até que uma dada condição seja satisfeita. Existem vários tipos de laços (*loops*) em Python. O mais comum e amplamente utilizado é o “for”. Normalmente o “for” é utilizado com objetos iteráveis, tais como listas e intervalos. Para simplificar, são utilizados dois exemplos: um com uma lista e outro com um intervalo. A Figura 1.21 demonstra um *loop* “for” percorrendo cada um dos elementos de uma lista. A condição de parada é o final da lista.

```
# Exemplo de utilização do laço de repetição for com lista
for i in [0, 1, 'oi', 3, 4.5, 'batata']:
  print(i)
```

Figura 1.21. Código Python para exemplificar um *loop* percorrendo uma lista

É interessante notar que não há diferença se a lista é de um determinado tipo único (como uma lista exclusivamente de número inteiros), ou se a lista tem valores mistos (como inteiros, *strings* e decimais). No exemplo da Figura 1.21, o *loop* percorre item por item passando o valor do item atual para a variável “i”. Já na Figura 1.22, pode-se notar um exemplo de utilização de laço de repetição “for” com um intervalo.

```
# Exemplo de utilização do laço de repetição for com intervalo
for i in range(5):
  print(i)
```

Figura 1.22. Código Python para exemplificar o uso de laços de repetição com intervalo

Os *loops* podem possibilitar, por exemplo, que os sistemas de recomendação percorram os valores advindos da base de dados para obter insumos, podendo sugerir

itens aos usuários. Sem a navegação dos itens realizada pelos laços seria pouco provável a viabilidade de implementar um bom algoritmo de recomendação.

1.6.8. Funções

Funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas várias vezes dentro de uma aplicação. Quando surge a necessidade de executar diversas vezes o mesmo código, criamos funções para que estas instruções não precisem ser repetidas atrapalhando a manutenção do código e causando dualidades. Assim, é possível "empacotar" trechos de códigos e utilizá-los em diversos lugares diferentes.

Uma função pode ser criada no Python utilizando a palavra reservada "def" seguida do nome da função e parâmetros desejados (caso existam). Após isso, deve ser criado o corpo da função que pode dispor de um retorno de valor ou não. A Figura 1.23 demonstra como criar e executar uma função sem retorno. Já a Figura 1.24 demonstra como criar e utilizar uma função genérica que retorna o salário semanal de um indivíduo e que pode ser utilizada em diversas outras partes de um programa.

```
def oi(nome):
   print('Olá',nome)
oi("Maria")
```

Figura 1.23. Código Python para exemplificar uma função simples

```
# Esta função calcula se a carga horária semanal passou de 40 horas e calcula o valor das horas extras. O resultado é o retorno do quanto o indivíduo deve receber.
def calcular_salario_semana(qtd_horas, valor_hora):
  horas = float(qtd_horas)
  taxa = float(valor_hora)
  if horas <= 40:
    salario=horas*taxa
  else:
    h_excd = horas - 40
    salario = 40*taxa+(h_excd*(1.5*taxa))
  return salario
mó
# 176 horas trabalhadas
salario = calcular_salario_semana(176, 10)
print("O valor que a pessoa precisa receber é: {}".format(salario))
```

Figura 1.24. Código Python para exemplificar uma função com retorno

As funções sem retorno, como a da Figura 1.23, não são muito usuais. Normalmente é esperado que, após o processamento realizado por uma função, seja retornado um resultado para que o fluxo continue a partir daquele valor (realizando outras operações ou exibindo algo ao usuário), assim como a função da Figura 1.24. É recomendado que as funções sejam mais genéricas possíveis para que o código possa ser reaproveitado ao máximo de vezes sem exigir alterações particulares para cada caso. Nesse contexto, é preciso criar algo genérico o suficiente para ser útil em várias partes do código, mas ao mesmo tempo com um grau de complexidade acessível que dê para realizar manutenção sem problemas maiores. Muitas funções possibilitam atender a diversos cenários, mas são extremamente complexas para serem entendidas. O mundo ideal é ter funções genéricas e simples de serem entendidas para futuras alterações e manutenções.

A linguagem Python dispõe ainda de diversas funções próprias prontas ao uso do programador. É possível, por exemplo, medir o tamanho de uma lista utilizando a função "len", além de indicar que todos os caracteres de uma *String* são minúsculos utilizando a função "lower". Existe uma gama de funções prontas para cada tipo de trabalho que se deseje realizar partindo desde formatação de texto até a análise de dados matemáticos complexos. A linguagem Python atualmente é uma das mais bem preparadas para atender tanto os programadores com demandas mercadológicas (criar sistemas, telas de autenticação, integrar com bancos de dados etc.) até pesquisadores que necessitam realizar cálculos em terabytes de dados.

1.7. *Hands-on*: Implementando um Algoritmo de Recomendação

A plataforma do Google Colab[3] foi escolhida como ambiente para a implementação de um algoritmo de recomendação simples, visando demonstrar de forma prática os conhecimentos anteriormente abordados nesse capítulo. Tal plataforma foi adotada por ser gratuita, on-line e por dispor de ambiente Python preparado com o necessário para a execução do código que é demonstrado nessa seção.

O Google Colab também conta com outros pontos positivos, como o fato do usuário poder salvar o *notebook* de códigos criados para utilizar e/ou estudar posteriormente. Além disso, também é possível acessá-lo de qualquer dispositivo que tenha acesso com a internet e navegador compatível. Por ser on-line, o ambiente do Google Colab proporciona que não seja necessário criar e configurar ambientes Python complexos para a implementação de algoritmos, simplificando, portanto, a empregabilidade do conhecimento abordado nessa seção e não exigindo a necessidade de conhecimentos sobre infraestrutura e sistemas operacionais para instalar e configurar o ambiente Python.

A presente seção aborda um exemplo de algoritmo de recomendação de filmes. É utilizada a base do MovieLens para exemplificar como um algoritmo de recomendação funciona. O objetivo dessa exemplificação é recomendar filmes que ainda não foram assistidos por um usuário de acordo com as avaliações da base de dados. Nessa base é possível selecionar avaliações de vários usuários u para diversos filmes f. Dessa forma, temos uma matriz u X f. Os filmes são avaliados considerando

[3] colab.research.google.com

uma escala de 1 até 5 (“avaliação 1 estrela” até “avaliação 5 estrelas”). Caso um usuário u não tenha avaliado um filme f, o valor dessa associação é 0.

Considerando que estamos utilizando a filtragem colaborativa como abordagem de recomendação, uma solução para prover recomendações de filmes f para um usuário u é analisar um conjunto de usuários S que são similares ao usuário u e que já assistiram ao filme f. A importação dessa base e a execução desses passos iniciais pode ser feita utilizando a biblioteca Python, Surprise. Através da utilização do Surprise, é possível carregar o dataset do MoviesLens, conforme demonstrado na Figura 1.25.

A partir do “ml-100k” é possível carregar um dataset com 100.000 registros de interação de usuários (linhas) com filmes (colunas) do MovieLens. O método “load_builtin()”, proveniente da biblioteca Surprise, permite a importação de 943 linhas e 1682 colunas deste *dataset*.

```
#instalação e importação do Surprise
#a exclamação antes do comando “pip” deve ser utilizada no caso de uso do Google Colab

!pip install surprise
from surprise import Dataset
from surprise import KNNBasic
#para carregar o dataset do MovieLens
dados = Dataset.load_builtin("ml-100k")
```

Figura 1.25. Código Python para importação da biblioteca Surprise e do *dataset* do MovieLens

A partir dessa matriz, é possível calcular a similaridade entre os filmes, ou seja, a abordagem de filtragem colaborativa chamada de baseado em item (*item-item*). Ao contrário da baseada em usuário (*user-based*), essa abordagem cria para cada filme um vetor de avaliações. Existem algumas implementações Python possíveis para calcular a similaridade entre dois vetores. Uma delas é a “cosine”. A similaridade “cosine” entre dois vetores pode ser facilmente calculada utilizando o “KNNBasic”. Dessa forma, é preciso indicar como propriedade da similaridade que não se trata de um “*user-based*” e de que se deseja implementar a “cosine”, conforme Figura 1.26.

```
opcoes_sim = {
    'name': 'cosine',
    'user_based': False
}
knn = KNNBasic(sim_options=opcoes_sim)
```

Figura 1.26. Código Python para declarar a similaridade cosine utilizando o KNNBasic

É possível definir também um conjunto de dados para treinamento utilizando toda a base carregada na variável “dados”. O “KNNBasic” representado no exemplo da Figura 1.26 pela variável “knn”, pode ser treinado utilizando o método “fit( )”, assim como demonstrado no código Python na Figura 1.27.

```
dados_treinamento = dados.build_full_trainset()
knn.fit(dados_treinamento)
```

Figura 1.27. Código Python para treinar o *dataset* utilizado

O principal beneficiado com as recomendações desse exemplo são aqueles usuários que ainda não assistiram algum filme da base. Para selecionar os pares desses usuários (usuário-filme) no conjunto treinado “trainingSet”, é possível utilizar o método “build_anti_testset()”. Com todos esses registros selecionados, é possível aplicar o método “test()” para se obter predições de avaliações.

Essas predições indicam (com base no conjunto treinado) que um usuário u que tenha avaliado positivamente alguns filmes de animação pode avaliar outros filmes semelhantes de animação com 5 estrelas, pode exemplo. Nesse contexto, há uma grande possibilidade que filmes de animação ainda não assistidos por esse usuário obtenham um índice maior de similaridade. Essa etapa está representada na Figura 1.28.

```
dados_teste = dados_treinamento.build_anti_testset()
predicoes = knn.test(dados_teste)
```

Figura 1.28. Código Python para gerar as predições da recomendação com base nos dados de treinamento

Com essas predições definidas, é possível ranquear e selecionar apenas as 4 predições mais bem pontuadas. Para isso, o método “get_top4_recomendacoes” é criado, conforme Figura 1.29.

```
from collections import defaultdict
 def get_top4_recomendacoes(predicoes, topN = 4):


   top_recomendacoes = defaultdict(list)
   for uid, iid, true_r, est, _ in predicoes:
      top_recomendacoes[uid].append((iid, est))
   for uid, avaliacoes_usuario in top_recomendacoes.items():
      avaliacoes_usuario.sort(key = lambda x: x[1], reverse = True)
      top_recomendacoes[uid] = avaliacoes_usuario[:topN]


   return top_recomendacoes
```

Figura 1.29. Código Python para declarar função de seleção apenas das quatro predições mais bem ranqueadas

Para isso, é preciso indicar que a variável "top_rcomendacoes" (a variável que armazena os resultados do método criado) corresponde a uma "defaultdict(list)", uma maneira mais simples da biblioteca "collections" de inicializar um dicionário. Nesse caso, o dicionário está sendo inicializado com uma coleção (uma lista). O retorno das predições é indicado pelos identificadores (*ids*) dos filmes. Para que seja possível compreender bem qual filme está no top-4 de determinado usuário, é possível utilizar um método[4] da biblioteca Surprise que cria uma relação (dicionário) dos nomes dos filmes e respectivos *ids*, conforme o código da Figura 1.30.

```
import os, io
def read_item_names():
    file_name = (os.path.expanduser('~') +
            '/.surprise_data/ml-100k/ml-100k/u.item')
   rid_to_name = {}
   with io.open(file_name, 'r', encoding='ISO-8859-1') as f:
      for line in f:
         line = line.split('|')
         rid_to_name[line[0]] = line[1]
    return rid_to_name
```

Figura 1.30. Código Python para declarar função de seleção de nome dos filmes do MovieLens

Fonte: Documentação da biblioteca Surprise[5]

Por fim, o código da Figura 1.31 demonstra como realizar a chamada dessas funções criadas para visualizar os resultados. O primeiro passo é utilizar a função "get_top4_recomendacoes" para selecionar apenas as 4 primeiras das predições geradas com o KNNBasic para cada aluno. Em seguida, é preciso selecionar os nomes desses filmes utilizando o método "read_item_names()". Como o retorno do método "get_top4_recomendacoes" é uma lista, é possível utilizar o método "items()" para iterar os valores da lista. Nesse caso, o objetivo é o de apenas associar o nome do filme para cada um dos "iid" (identificador do item, ou seja, o filme retornado no método).

```
top4_recomendacoes = get_top4_recomendacoes(predicoes)
rid_to_name = read_item_names()
for uid, avaliacoes_usuario in top4_recomendacoes.items():
   print(uid, [rid_to_name[iid] for (iid, _) in avaliacoes_usuario])
```

Figura 1.31. Código Python para exibir recomendação para os usuários sem classificações do *dataset*

[4] Disponível em https://surprise.readthedocs.io/en/stable/FAQ.html

[5] Disponível em https://surprise.readthedocs.io/en/stable/FAQ.html

Após execução do código Python da Figura 1.31 no Python, é retornado imediatamente na tela uma lista com o identificador do usuário (id), seguido pelo nome dos quatro filmes mais similares de acordo com o algoritmo anteriormente implementado. Uma parte desse retorno pode ser visualizado na Figura 1.32, que exibe os resultados para os usuários de *ids* 196,186, 22, 166, 298 e 115. É possível perceber que cada nome de filme vem acompanhado também do ano de lançamento entre parênteses.

```
196 ['Very Natural Thing, A (1974)', 'Walk in the Sun, A (1945)', 'War at Home, The (1996)', 'Sunchaser, The (1996)']

186 ['Mamma Roma (1962)', 'Conspiracy Theory (1997)', 'Toy Story (1995)', 'MatchMaker, The (1997)']

22 ['Entertaining Angels: The Dorothy Day Story (1996)', 'King of New York (1990)', 'Usual Suspects, The (1995)', "One Flew Over the Cuckoo's Nest (1975)"]

244 ['Other Voices, Other Rooms (1997)', 'Big Bang Theory, The (1994)', 'Godfather, The (1972)', 'Casablanca (1942)']

166 ['Mamma Roma (1962)', 'Delta of Venus (1994)', 'Carmen Miranda: Bananas Is My Business (1994)', 'Marlene Dietrich: Shadow and Light (1996) ']

298 ['North by Northwest (1959)', 'Pinocchio (1940)', 'Amadeus (1984)', 'When Harry Met Sally... (1989)']

115 ['2001: A Space Odyssey (1968)', 'Clockwork Orange, A (1971)', 'Three Colors: White (1994)', 'Leaving Las Vegas (1995)']
```

Figura 1.32. Extrato de resultados da recomendação de filmes utilizando biblioteca Surprise

1.8. Principais Desafios e Oportunidades de Pesquisa

Apesar dos algoritmos de recomendação serem amplamente utilizados nos últimos anos, ainda existem diversos desafios em aberto. A privacidade de dados abordada na Seção 1.5 é um aspecto que deve ser considerado em qualquer solução que manipule dados de usuários. Além dessas questões abordadas, pode-se mencionar alguns desafios:

- Falta de dados: não é sempre que usuários possuem dados a serem extraídos das plataformas para que seja feita a recomendação. Em uma aplicação de venda de imóveis, por exemplo, podemos considerar que seria aceitável se a aplicação não dispusesse de muitos dados de muitos clientes, uma vez que a compra de uma casa é algo pouco frequente. Para solucionar esse desafio, os sistemas de recomendação podem implementar técnicas baseadas em conhecimento que incluem uma base de conhecimento do usuário para apoiar na recomendação [Aggarwal 2016].
- Dificuldade para mostrar novos dados: em uma recomendação colaborativa de filmes, como a do nosso exemplo, é preciso considerar que pode existir uma dificuldade de incluir no resultado final da recomendação determinados filmes que são pouco classificados por usuários. Isso ocorre devido ao fato de que a classificação de usuários é um critério importante no cálculo de similaridade

entre itens. Se há ausência de classificações, pode existir consequentemente um baixo índice de recomendação.

- *Cold-start*: ainda há uma dificuldade em recomendar itens para usuários que não possuem muito tempo de utilização das plataformas. Logo, esses usuários possuem poucos dados registrados, como o de vizinhos (no caso da filtragem colaborativa) ou de históricos (no caso da baseada em conteúdo) para que a recomendação tenha mais chances de acerto.
- Itens imprevisíveis: alguns itens não possuem uma classificação totalmente em comum para todos os usuários. Um exemplo seria em recomendação de músicas. Algumas músicas são adoradas por muitos usuários e odiadas por diversos outros. Isso faz com que as soluções de recomendação tenham dificuldade de identificar se essa música pode ser um item relevante a ser considerado no processo de recomendação.

Os problemas mencionados são amplamente envolvidos em tópicos de pesquisa sobre sistemas de recomendação. Surgem possibilidades de inclusão de técnicas de mineração de dados (como em [Campos et al. 2019]), Linked Open Data (como em [Noia e Ostuni 2015]), redes sociais [Logesh et al. 2018], bem como estudos que abordem a perspectiva de Ecossistemas de Software (ECOS) para melhor compressão das interações do software, facilitando a reutilização (conforme adotado em [Campos et al. 2018] e/ou possibilitando o compartilhamento de soluções de recomendação, permitindo a integração com outros sistemas de informação (a exemplo de [Abdalla et al. 2018]).

1.9. Conclusão

Com o avanço da tecnologia, os algoritmos de recomendação têm sido utilizados cada vez mais, e em diversos cenários. Seja utilizando dados de diversos usuários ou apenas dados históricos de um usuário interessado, esses algoritmos permitem solucionar problemas, otimizar os processos de busca e recuperação da informação e consequentemente atribuem valor para as aplicações que os utilizam. Apesar da fácil compreensão do objetivo final de um algoritmo de recomendação, o processo em si pode ser complexo e pode utilizar ainda outras técnicas, como a de mineração de dados, para auxiliar no resultado final.

Uma preocupação existente na utilização desses algoritmos atualmente é a privacidade de dados dos usuários. Isso demanda uma necessidade em reavaliar quais dados são utilizados para o processo de recomendação de um usuário, quais dados de um usuário são utilizados para os processos de recomendação de outros usuários, mas também políticas públicas de privacidade de dados que amparem usuários de diversos websites ou qualquer solução que possua extração e uso de dados de usuários.

Esse capítulo apresentou os principais conceitos dos algoritmos de recomendação, ressaltando teorias da literatura que demonstram o funcionamento e o comportamento dos diversos tipos de soluções de recomendação. Entendemos ser essencial que os conhecimentos de linguagem de programação sejam introduzidos para que um interessado em implementar um algoritmo de recomendação possa compreender seu funcionamento em sua totalidade. Nesse contexto, adotamos o Python como linguagem de programação e demostramos as principais operações através do Google Colab.

A implementação do algoritmo de recomendação demonstrada nesse capítulo pode ser totalmente adotada através de outra linguagem de programação, desde que respeitados as particularidades de cada linguagem. Os resultados da recomendação exibem como essa técnica pode auxiliar na recuperação de uma informação próxima daquilo que determinado usuário busca, sendo assim, portanto, um recurso importante para as técnicas de busca e recuperação da informação.

Referências

Abdalla, A., Ströele, V., Campos, F., David, J. M. N. and Braga, R. (2018). Plataforma de Ecossistema de Software para Sistemas de Recomendação. In *Anais do XIV Simpósio Brasileiro de Sistemas de Informação*. . SBC.

Aggarwal, C. C. (2016). *Recommender systems*. Cham: Springer International Publishing.

Armknecht, F. and Strufe, T. (2011). An Efficient Distributed Privacy-preserving Recommendation System. [IEEE, Ed.]In *2011 The 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop.* . https://researcher.ibm.com/researcher/view.

Baeza-Yates, R. and Ribeiro-Neto, B. (2013). *Recuperação de Informação - 2ed: Conceitos e Tecnologia das Máquinas de Busca*. Bookman Editora.

Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, v. 40, n. 3, p. 66–72.

Campos, R., Santos, R. P. and Oliveira, J. (2018). Web-Based Recommendation System Architecture for Knowledge Reuse in MOOCs Ecosystems. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. . IEEE. https://ieeexplore.ieee.org/document/8424707/.

Campos, R., Santos, R. P. and Oliveira, J. (2019). A Recommendation System Enhanced by Topic Modeling for Knowledge Reuse in MOOCs Ecosystems. *Reuse in Intelligent Systems*. Newcastle upon Tyne, UK: Cambridge Scholars Publishing (in press). .

Cazella, S. C., Chagas, I. C. Das, Barbosa, J. L. V. and Reategui, E. B. (2008). Um modelo para recomendação de artigos acadêmicos baseado em filtragem colaborativa aplicado à ambientes móveis. *RENOTE: revista novas tecnologias na educação [recurso eletrônico]. Porto Alegre, RS*,

Cazella, S. C., Drumm, J. V. and Barbosa, J. L. V (2010). Um serviço para recomendação de artigos científicos baseado em filtragem de conteúdo aplicado a dispositivos móveis. *RENOTE*, v. 8, n. 3.

Cazella, S. C., Nunes, M. and Reategui, E. (2010). A Ciência da Opinião: Estado da arte em Sistemas de Recomendação. *André Ponce de Leon F. de Carvalho; Tomasz Kowaltowski..(Org.). Jornada de Atualização de Informática-JAI*, p. 161–216.

Costa, E., Aguiar, J. and Magalhães, J. (2013). Sistemas de Recomendação de Recursos Educacionais: conceitos, técnicas e aplicações. *Jornada de Atualização em Informática na Educação*, v. 1, n. 1.

Desrosiers, C. and Karypis, G. (2011). A Comprehensive Survey of Neighborhood-based Recommendation Methods. *Recommender Systems Handbook*. Boston, MA:

Springer US. p. 107–144.

Do, M.-P. T., Nguyen, D. V. and Nguyen, L. (2010). Model-based Approach for Collaborative Filtering. *Proceedings of The 6th International Conference on Information Technology for Education (IT@EDU2010)*, n. August 2010, p. 217–225.

Feng, P., Zhu, H., Liu, Y., Chen, Y. and Zheng, Q. (2018). Differential Privacy Protection Recommendation Algorithm Based on Student Learning Behavior. In *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*.

Grossman, D. A. and Frieder, O. (2004). *Information Retrieval: Algorithms and Heuristics*. Springer.

Herlocker, J. L., Konstan, J. A. and Riedl, J. (2000). Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*.

Jiang, J.-Y., Li, C.-T. and Lin, S.-D. (2019). Towards a More Reliable Privacy-preserving Recommender System. *Journal of Information Sciences*, v. 482, p. 248--265.

Jones, K. S. (1997). *Readings in Information Retrieval*. Morgan Kaufman.

Konstan, J. A., Miller, B. N., Maltz, D., et al. (1997). GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, v. 40, n. 3, p. 77--87.

Koren, Y. (2008). Factorization meets the neighborhood. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*. . ACM Press. http://doi.acm.org/10.1145/.

Logesh, R., Subramaniyaswamy, V. and Vijayakumar, V. (2018). A personalised travel recommender system utilising social network profile and accurate GPS data. *Electronic Government, an International Journal*, v. 14, n. 1, p. 90–113.

Mcsherry, F. and Mironov, I. (2009). Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Motta, C. L. R. Da, Garcia, A. C. B., Vivacqua, A. S., Santoro, F. M. and Sampaio, J. O. (2011). Sistemas de recomendação. *Pimentel, M.; Fuks, H."Sistemas colaborativos". Rio de Janeiro: Elsevier*,

Noia, T. Di and Ostuni, V. C. (2015). Recommender Systems and Linked Open Data. *Reasoning Web International Summer School*. Springer. http://www.pandora.com/, [accessed on Apr 13].

Qi, L., Xiang, H., Dou, W., et al. (7 sep 2017). Privacy-Preserving Distributed Service Recommendation Based on Locality-Sensitive Hashing. In *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*. . Institute of Electrical and Electronics Engineers Inc.

Ricci, F., Rokach, L., Shapira, B. and Kantor, P. B. (2015). *Recommender systems handbook*. Springer.

Santos, L. C. de M. Dos and Bräscher, M. (31 dec 2017). Uso de ontologia na recuperação da informação em acervos digitais de jornais. *Informação & Informação*, v. 22, n. 3, p. 346.

Shardanand, U. and Maes, P. (1995). Social Information Filtering: Algorithms for Automating \Word of Mouth". *Chi*. Citeseer. v. 95p. 210--217.

Wang, C., Zheng, Y., Jiang, J. and Ren, K. (1 feb 2018). Toward Privacy-Preserving Personalized Recommendation Services. *Engineering*, v. 4, n. 1, p. 21–28.

Sobre os Autores

Leonardo Herdy Marinho

Mestrando no Programa de Pós-Graduação em Informática (PPGI) da Universidade Federal do Rio de Janeiro (UFRJ) na área de Sistemas de Informação, com um tema de pesquisa focado em investigar como os sistemas de recomendação podem apoiar os ecossistemas de startups na geração de parcerias. Profissional com 5 anos de experiência como analista e desenvolvedor de sistemas. Graduado em Análise e Desenvolvimento de Sistemas, possui experiência em projetos com PHP, Laravel, Javascript, NodeJs, Angular, Ionic, Dart, Flutter, Aqueduct, Django e Python.

Rodrigo Campos

Mestrando no Programa de Pós-Graduação em Informática (PPGI) da Universidade Federal do Rio de Janeiro (UFRJ) na área de Sistemas de Informação, com um tema de pesquisa focado em investigar como os sistemas de recomendação podem apoiar os ecossistemas MOOCs na reutilização do conhecimento. Profissional com 4 anos de experiência como desenvolvedor de sistemas em empresas governamentais brasileiras, como a Marinha do Brasil e atualmente o Instituto Federal de Educação, Ciência e Tecnologia do Rio de Janeiro (IFRJ). Graduado em Análise e Desenvolvimento de Sistemas, possui experiência em projetos com JavaEE, JSP, Hibernate e SpringMVC.

Rodrigo Pereira dos Santos

Professor Adjunto do Departamento de Informática Aplicada (DIA) e membro efetivo do Programa de Pós-Graduação em Informática (PPGI) da Universidade Federal do Estado do Rio de Janeiro (UNIRIO), onde atualmente é Coordenador do Curso de Mestrado. Atuou como pesquisador visitante na University College London (BEX/CAPES, 2014-2015). Atuou como consultor em projetos de pesquisa e desenvolvimento em engenharia de sistemas na indústria nacional pela Fundação Coppetec (2008-2017). É editor-chefe da iSys: Revista Brasileira de Sistemas de Informação. É membro da Sociedade Brasileira de Computação (SBC) desde 2006 e Coordenador do Comitê Gestor da Comissão Especial de Sistemas de Informação (CE-SI) da SBC. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software e Sistemas de Informação. Seus principais campos de atuação são Engenharia de Sistemas Complexos (especialmente ecossistemas de software e sistemas-de-sistemas) e Educação em Engenharia de Software. Foi coordenador científico de mais de 20 eventos (simpósios, trilhas e workshops) no Brasil e no exterior e proferiu comunicações (palestras, minicursos e tutoriais) em mais de 20 eventos nacionais.

Mônica Ferreira da Silva

Professora do Programa de Pós-graduação em Informática (PPGI) da Universidade Federal do Rio de Janeiro (UFRJ). Doutora em Administração pelo COPPEAD/UFRJ (2006). Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ (1998). Especialização em Gerência de Projetos pelo NCE/UFRJ (2001). Graduação em Informática pela Universidade Federal do Rio de Janeiro (UFRJ) concluído com grau Cum Laude de dignidade acadêmica (1988). Tem experiência nas áreas de Ciência da Computação e Administração, com ênfase em Gestão da Tecnologia da Informação. Atuando principalmente nos seguintes temas: estratégia e sistemas de informação, metodologia de pesquisa científica e adoção de tecnologia.

Jonice Oliveira

Short Bio: Profa. Jonice Oliveira obteve o seu doutorado em 2007 na área de Engenharia de Sistemas e Computação, ênfase em Banco de Dados, pela COPPE/UFRJ. Durante o seu doutorado recebeu o prêmio IBM Ph.D. Fellowship Award. Na mesma instituição realizou o seu Pós-Doutorado, concluindo-o em 2008. Desde 2009 é professora do Departamento de Ciência da Computação da UFRJ e atualmente é coordenadora do Programa de Pós-Graduação em Informática (PPGI-UFRJ). Coordena o Laboratório CORES (Laboratório de Computação Social e Análise de Redes Sociais), que conduz pesquisas multidisciplinares para o entendimento, simulação e fomento às interações sociais. Sua principal área de pesquisa é Computação Social, mais especificamente nos temas de Análise de Redes Sociais, Big Social Data, Suporte à Decisão e Recomendação. Possui uma larga experiência em tais áreas, com mais de centenas de artigos publicados, dezenas de orientações e envolvimento (como membro ou como líder) em projetos de pesquisas nacionais e internacionais. Maiores detalhes em: http://www.joniceoliveira.net/.