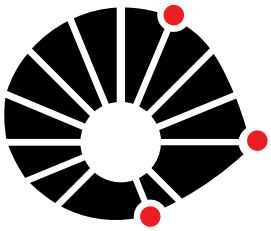

UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS

Instituto de Matemática, Estatística e Computação Científica

CAMILA JARDIM CAVALCANTE BONILLA

Uma introdução aos sistemas de recomendação: modelos matemáticos, algoritmos e aplicações

Campinas

2020

Camila Jardim Cavalcante Bonilla

Uma introdução aos sistemas de recomendação: modelos matemáticos, algoritmos e aplicações

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestra em Matemática Aplicada e Computacional.

Orientador: Cristiano Torezzan

Este exemplar corresponde à versão final da Dissertação defendida pela aluna Camila Jardim Cavalcante Bonilla e orientada pelo Prof. Dr. Cristiano Torezzan.

Campinas

2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

B641i

Bonilla, Camila Jardim Cavalcante, 1992-
Uma introdução aos sistemas de recomendação : modelos matemáticos, algoritmos e aplicações / Camila Jardim Cavalcante Bonilla. – Campinas, SP : [s.n.], 2020.

Orientador: Cristiano Torezzan.
Dissertação (mestrado profissional) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Sistemas de recomendação (Filtragem da informação). 2. Aprendizado de máquina. 3. Redução de dimensionalidade (Estatística). 4. Método K-vizinho mais próximo. 5. Árvores de decisão. 6. Redes neurais (Computação). I. Torezzan, Cristiano, 1976-. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: An introduction to recommendation systems : mathematical models, algorithms and applications
Palavras-chave em inglês:
Recommender systems (Information filtering)
Machine learning
Dimension reduction (Statistics)
K-nearest neighbor
Decision trees
Neural networks (Computer science)
Área de concentração: Matemática Aplicada e Computacional
Titulação: Mestra em Matemática Aplicada e Computacional
Banca examinadora:
Cristiano Torezzan [Orientador]
Washington Alves de Oliveira
Lucas Garcia Pedroso
Data de defesa: 15-12-2020
Programa de Pós-Graduação: Matemática Aplicada e Computacional

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-0563-5438
- Currículo Lattes do autor: http://lattes.cnpq.br/9408921263160458

Dissertação de Mestrado Profissional defendida em 15 de dezembro de 2020 e aprovada pela banca examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). CRISTIANO TOREZZAN

Prof(a). Dr(a). WASHINGTON ALVES DE OLIVEIRA

Prof(a). Dr(a). LUCAS GARCIA PEDROSO

A Ata da Defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria de Pós-Graduação do Instituto de Matemática, Estatística e Computação Científica.

Ao meu Deus, Aquele que por mim tudo executa.
Salmos 57:2

Agradecimentos

Existe um provérbio africano que diz que “é preciso um vilarejo inteiro para criar uma criança”. Poderíamos facilmente adaptá-lo e dizer que é preciso um vilarejo inteiro para formar um mestre. Nesses anos de mestrado foram muitas as pessoas envolvidas e que, de uma maneira ou outra, ofereceram suporte para que eu pudesse prosseguir em alcançar meu objetivo.

Em primeiro lugar agradeço ao meu Deus, cuja graça esteve todo o tempo sobre mim e que confirmou a obra das minhas mãos.

Agradeço ao meu esposo, Josué, meu cuidador e incentivador. Sei que grande parte do que sou e do que tenho construído só é possível porque tenho você ao meu lado e o seu amor todos os dias. Essa conquista também é sua.

Agradeço à minha família em todo o seu tamanho: meu pai, minha mãe, meus irmãos, minha sogra, meu sogro e meus cunhados. Obrigada pelo amor, compreensão e incentivo. Agradeço em especial ao meu pai, em quem eu vejo todos os dias o amor pelo conhecimento. Obrigada pelo seu apoio sempre fiel e sempre presente.

Agradeço ao meu orientador Prof. Dr. Cristiano Torezzan. Obrigada pelo tempo, paciência, incentivo e interesse genuíno em meu desenvolvimento acadêmico. Me sinto honrada em ter trabalhado de maneira mais próxima com alguém a quem tanto admiro.

Agradeço a todo o corpo de professores e coordenadores do programa do Mestrado Profissional em Matemática Aplicada e Computacional do IMECC, em especial a Prof. Dr. Sueli Costa, ao Prof. Dr. Simão Stelmastchuk, ao Prof. Dr. João Frederico Meyer (Joni) e a Prof. Dr. Priscila Rampazzo por seu empenho e esmero na função de professores. Agradeço também aos funcionários do IMECC por serem sempre tão solícitos e gentis, sobretudo a equipe da secretaria de pós-graduação e da biblioteca.

Agradeço as equipes e aos meus superiores nas empresas onde trabalhei durante o período do mestrado, Falconi, Mercado Livre e PagSeguro, pela compreensão e apoio durante os momentos em que era preciso me ausentar para participar das aulas na UNICAMP. Agradeço especialmente a dois grandes amigos que estiveram muito próximos durante boa parte da jornada, Porto e Wander. Que venham ainda muitos anos de conquistas juntos.

Aos meus amigos que viraram família durante o período do mestrado, vai meu agradecimento já cheio de saudades. Makson, Andrey, Enio e Suellen, a amizade de vocês foi um dos grandes fatores de sucesso dessa empreitada (e as muitas horas de desespero pré-prova estudando no “predinho da pós”, só para comemorarmos - ou nos consolarmos -

com pizza da Vila Ré e sorvete do Machuchos no dia seguinte).

Agradeço também aos professores que compõem a banca pela avaliação e comentários que muito contribuíram para o refinamento do trabalho.

Aos que fizeram parte dessa conquista mas que, por algum motivo, não nomeei nos agradecimentos acima, minha gratidão sincera.

Resumo

Sistemas de recomendação têm sido amplamente utilizados, sobretudo no ramo do comércio eletrônico, para compilar informações sobre as preferências dos usuários, com o intuito de sugerir itens de maneira personalizada e facilitar a busca por produtos e serviços. Tais sistemas são baseados em modelos matemáticos e algoritmos computacionais, que permitem processar um grande volume de dados e automatizar as recomendações. Neste trabalho apresenta-se uma revisão sobre as principais abordagens utilizadas na modelagem dos sistemas de recomendação, denominadas sistemas de filtragem colaborativa e sistemas de filtragem baseada em conteúdo, bem como as métricas de avaliação utilizadas. O trabalho inclui uma síntese sobre a história dos sistemas de recomendação e exemplos de casos de uso em grandes empresas de tecnologia. São discutidos com mais detalhes os algoritmos de regressão linear, redução de dimensionalidade de matrizes, K-vizinhos mais próximos, árvores de decisão e redes neurais artificiais. Por fim, apresenta-se a aplicação dos algoritmos de K-vizinhos mais próximos, redução de dimensionalidade de matrizes e redes neurais artificiais na construção de um sistema de recomendação para uma loja online de vestuário.

Palavras-chave: Sistemas de recomendação, aprendizado de máquina, redução de dimensionalidade de matrizes, K-vizinhos mais próximos, árvores de decisão, redes neurais artificiais.

Abstract

Recommendation systems have been widely used, especially in e-commerce, in order to compile information on users' preferences and suggest items in a personalized manner, facilitating the users' search for products and services. These systems are based on mathematical models and computational algorithms that enable the processing of a large volume of data and the automation of recommendations. This text reviews the main methods used in modeling recommendation systems, that is, collaborative filtering systems and content based filtering systems, as well as some of the evaluation metrics used. A synthesis on the history of recommendation systems is included, along with examples of their use in big technology companies. The algorithms discussed in greater detail are linear regression, matrix dimensionality reduction, K-nearest neighbors, decision trees, and artificial neural networks. Finally, applications of the algorithms K-nearest neighbors, matrix dimensionality reduction, and artificial neural networks in the construction of a recommendation system for a clothing online store are presented.

Keywords: Recommendation systems, machine learning, matrix dimensionality reduction, K-nearest neighbors, decision trees, artificial neural networks.

Lista de ilustrações

Lista de tabelas

Lista de algoritmos

Sumário

Introdução

O avanço das comunicações digitais e a integração das cadeias de suprimento propiciaram um grande avanço no comércio digital e o oferecimento de uma enorme variedade de itens de forma *online*. Segundo dados de abril de 2019 (SCRAPEHERO, 2019), o site de comércio eletrônico *Amazon.com* possuía quase 120 milhões de produtos em sua plataforma ao redor do mundo. Dentre as muitas categorias, a maior era a de "Literatura", com 44,2 milhões de exemplares. A categoria com o menor número de itens era a de "Ferramentas e melhorias domésticas", com 3,7 milhões de produtos - ou seja, caso um potencial cliente queira comprar um kit de ferramentas, se deparará com uma escolha entre milhões.

Ainda que mecanismos de busca ajudem a aliviar esse problema ao permitir que o usuário especifique as características do produto no qual tem interesse, negligenciam a enorme oportunidade existente através do mapeamento dos interesses do cliente e a personalização da oferta de produtos.

De forma a atender a necessidade de personalização de ofertas e mitigar o chamado *information overload* (excesso de informação, que tende a impactar negativamente o processo de tomada de decisão), foram criados os chamados "sistemas de recomendação". Essa tecnologia permite extrair as informações mais relevantes de uma grande e dinâmica base de dados com base nas características dos produtos e no perfil dos usuários.

Além de recomendações de produtos, como no caso da *Amazon* citado acima, sistemas de recomendação são utilizados também para indicar ao usuário potenciais conteúdos de interesse. Por exemplo, o jornal americano *The New York Times* utiliza um sistema de recomendação para sugerir novos artigos aos seus leitores. Plataformas de mídia como *Netflix* e *Spotify* utilizam esses sistemas para gerar recomendações de filmes e músicas. Um caso interessante é também o da plataforma de músicas *Pandora*, que ficou conhecida por seu *"Music Genome Project"*. O projeto, iniciado há mais de uma década, analisa as ondas sonoras de músicas a fim de encontrar características como timbre e estilo musical. Com uma base de dados onde cada música tem ao final 450 atributos analisados, a iniciativa promete personalizar a experiência do usuário (PANDORA, 2019). Até mesmo ferramentas de busca como *Google* e *Yahoo* utilizam sistemas de recomendação para classificar os resultados de acordo com interesses e buscas anteriormente realizadas pelo usuário (KANE, 2018).

Mas afinal, como funcionam os sistemas de recomendação? Quais são os principais métodos e modelos conceituais utilizados? Este trabalho busca investigar essas questões e trazer algumas respostas sob a ótica da matemática aplicada. Assim, o ob-

jetivo desse texto é abordar certas aplicações de sistemas de recomendação com ênfase na compreensão de alguns dos modelos matemáticos e algoritmos computacionais mais utilizados. No Capítulo 1 apresenta-se uma visão geral sobre sistemas de recomendação, iniciando pela revisão histórica, casos de uso em diferentes segmentos de mercado, os modelos de sistemas mais utilizados e métricas de avalição de qualidade para sistemas de recomendação. As principais referências utilizadas nesse capítulo foram (AGGARWAL, 2016) e (KANE, 2018).

No Capítulo 2 são abordados alguns dos principais algoritmos empregados na construção de sistemas de recomendação, oferecendo ao leitor uma breve introdução sobre conceitos básicos no campo de aprendizado de máquina e seguindo para apresentar alguns entre os algoritmos mais utilizados na criação de sistemas de recomendação. Aborda-se o algoritmo do método dos K-vizinhos mais próximos, redução de dimensionalidade de matrizes, árvores de decisão e redes neurais artificiais. As principais referências utilizadas foram (AGGARWAL, 2016), (MARSLAND, 2009), (KANE, 2018), (HASTIE; TIBSHIRANI; FRIEDMAN, 2017), (GOODFELLOW; BENGIO; COURVILLE, 2016) e (STONE, 2020).

No Capítulo 3 apresenta-se o estudo de quatro sistemas de recomendação criados a partir de alguns dos algoritmos discutidos no segundo capítulo, especificamente os algoritmos dos K-vizinhos mais próximos, redução de dimensionalidade de matrizes e redes neurais artificiais. Esses sistemas foram utilizados como estudos de caso, com o intuito de testar os algoritmos estudados e as implementações que foram feitas, na análise de dados reais, obtidos a partir de uma base pública. No Capítulo 4, são apresentadas as conclusões e algumas perspectivas futuras.

Vale salientar que o estudo de sistemas de recomendação tem ramificações e especificidades que transcendem o escopo dessa dissertação. No entanto, a expectativa é que esse texto possa servir de inspiração para estudantes de matemática, estatística, engenharia ou áreas afins que estiverem interessados em alguns aspectos estruturantes que são utilizados para modelar tais sistemas.

1 Sistemas de recomendação: Noções Básicas

1.1 Revisão histórica

Os primeiros sistemas de recomendação foram criados no início dos anos 90, como uma resposta ao crescimento do volume de conteúdo disponível na internet. Um dos primeiros artigos publicados no assunto, desenvolvido no centro de pesquisa da Xerox, em Palo Alto, em 1992, analisava os problemas decorrentes do uso crescente de comunicações eletrônicas, que terminavam por sobrecarregar o usuário com documentos desnecessários ou de pouco interesse. No sistema chamado *Tapestry* os usuários podiam listar critérios de filtragem de conteúdos, como palavras-chave e outros usuários com os quais compartilhavam interesses. Por exemplo, um usuário poderia solicitar algo como "todos os artigos com a palavra *fintech* que contém comentários do Ricardo Silva" (GOLDBERG et al., 1992).

Dois anos depois, o projeto *GroupLens* apresentado na *Association for Computing Machinery 1994 Conference on Computer Supported Cooperative Work* utilizava um sistema de recomendação de *Collaborative Filtering* (ou "Filtro Colaborativo" na tradução literal) que permitia aos usuários da rede *Usenet* encontrar artigos de interesse em meio a um grande volume de informação (RESNICK et al., 1994). O sistema *GroupLens*, assim como o *Tapestry*, se baseava nas interações de pessoas com opiniões semelhantes para criar sugestões personalizadas. A diferença entre os dois sistemas se dava uma vez que o *GroupLens* gerava as recomendações fundamentado nas avaliações de outros usuários, e não exigia que o usuário inicial especificasse com quais outros usuários possuía similaridade.

Em 1995 o sistema *Ringo* foi apresentado na *CHI 1995 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* como um *Social Information Filtering* que explorava a similaridade entre os interesses musicais de vários usuários para recomendar ou desaconselhar músicas (SHARDANAND; MAES, 1995).

A partir de 1996, a popularidade de sistemas de recomendação cresceu rapidamente, tanto em pesquisas acadêmicas como comercialmente, e várias empresas se especializaram em construir e comercializar esse tipo de sistema (KONSTAN; RIEDL, 2012).

1.2 Casos de destaque no uso de sistemas de recomendação

Nesta seção são discutidos três casos que estão entre os mais emblemáticos no uso de sistemas de recomendação em cenários comerciais.

1.2.1 Uso em comércio eletrônico: *Amazon.com*

A plataforma de comércio eletrônico *Amazon.com* foi uma das pioneiras no uso de sistemas de recomendação de forma comercial. As informações fornecidas pelo usuário e utilizadas na construção do sistema podem ser coletadas de maneira explícita e de maneira implícita. Dentre as informações explícitas podem-se considerar as notas, em uma escala de 1 a 5, que o usuário atribui aos produtos que comprou. De maneira implícita a empresa consegue coletar informações sobre o comportamento do usuário na plataforma, como por exemplo as categorias dos produtos mais buscados e comprados pelo usuário (AGGARWAL, 2016).

Uma vez feito o login do usuário em sua conta *Amazon*, as recomendações são exibidas logo na primeira página. Na parte superior da Figura 1 são apresentadas recomendações baseadas nas últimas pesquisas e compras do usuário. Mais abaixo são vistas recomendações de livros comprados por outros usuários que possuem características similares ao usuário em questão.

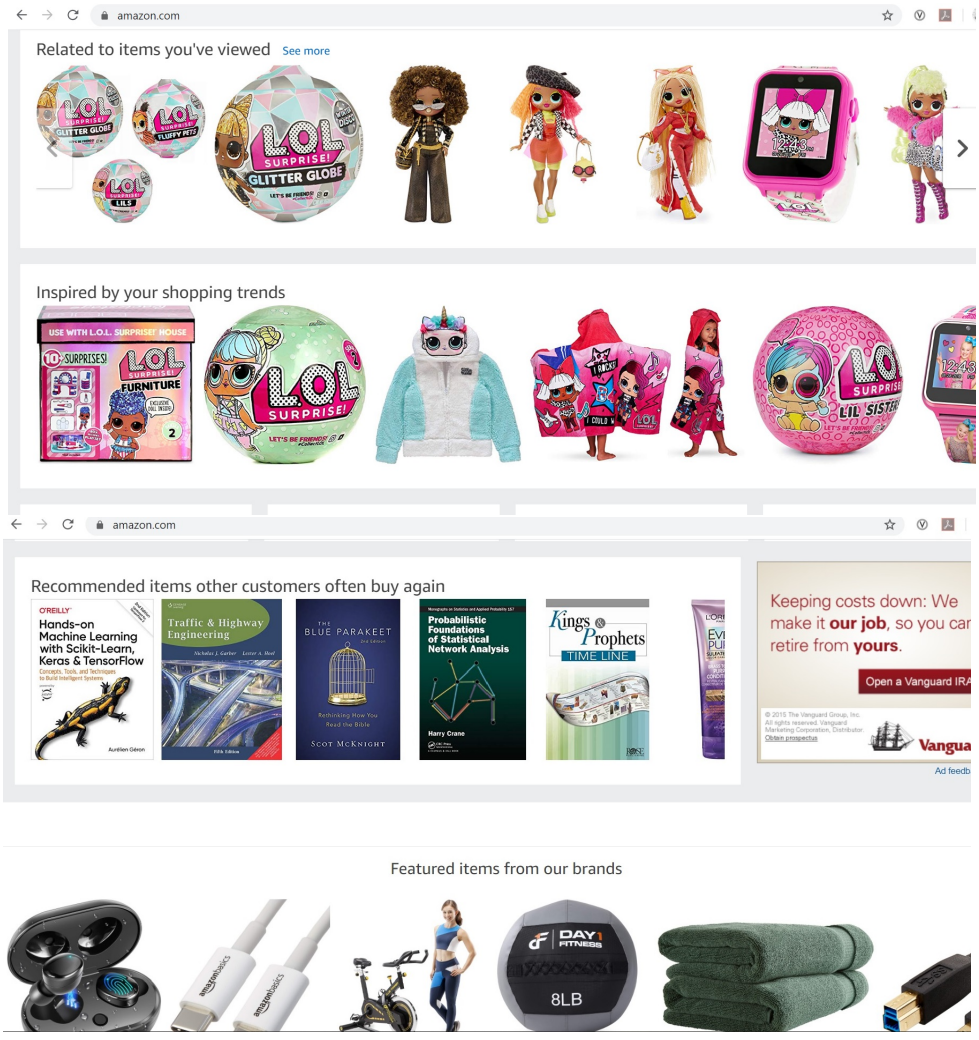

Figura 1 – Exemplo de recomendações de produtos na plataforma *Amazon.com*. Fonte: autora.

Sistemas de recomendação geram grande parte da receita da *Amazon*. Um artigo de 2013 elaborado pela consultoria estratégica *McKinsey & Company* afirma que 35% dos produtos vendidos no site são recomendações geradas pelos sistemas da empresa (MACKENZIE; MEYER; NOBLE, 2013).

1.2.2 Uso em plataformas de *streaming*: *Netflix*

Como uma plataforma de *streaming* de filmes e séries, a *Netflix* oferece ao usuário recomendações baseadas tanto no comportamento do usuário, a exemplo dos últimos itens assistidos ou melhores avaliados, quanto no comportamento de outros usuários da plataforma, como as melhores avaliações ou temas mais populares. Abaixo é possível ver um conjunto de recomendações baseadas no fato de que o usuário anteriormente havia assistido ao seriado *"The Witcher"*.

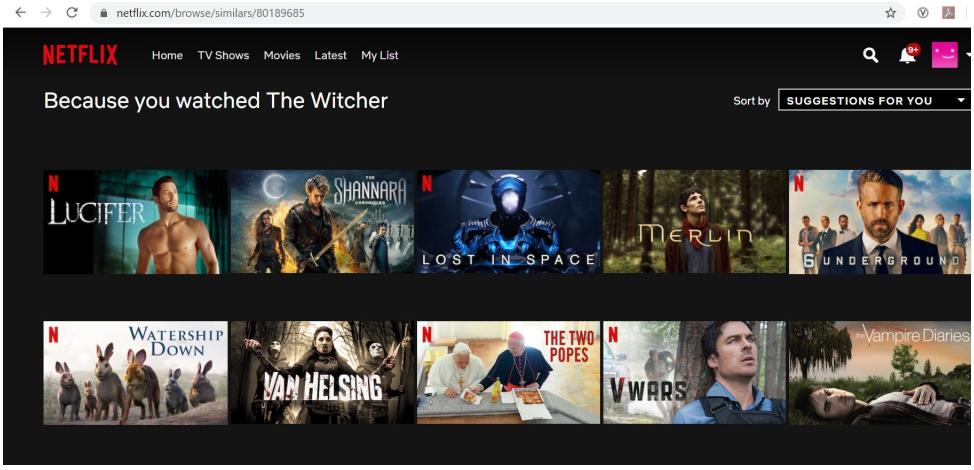

Figura 2 – Exemplo de recomendações de filmes e séries na plataforma de *streaming Netflix*. Fonte: autora.

Os sistemas de recomendação da *Netflix* se tornaram especialmente populares através do *Netflix Prize Contest*, uma competição patrocinada pela empresa a fim de fomentar a discussão e desenvolvimento de sistemas de recomendação com filtragem colaborativa (mais detalhes sobre sistemas de filtragem colaborativa serão dados nas próximas seções). O objetivo da competição era desenvolver um algoritmo que pudesse predizer as notas dadas por certos usuários a certos filmes. Os prêmios eram entregues de acordo com as melhorias apresentadas na comparação entre o algoritmo padrão da *Netflix* e o algoritmo apresentado, ou de acordo com as melhorias quando o algoritmo apresentado era comparado ao algoritmo melhor colocado até o momento. É reconhecido que o *Netflix Prize Contest* promoveu contribuições significantes para a pesquisa em sistemas de recomendação (AGGARWAL, 2016).

1.2.3 Uso em redes sociais: *Facebook*

Como outras redes sociais, o *Facebook* exibe ao usuário sugestões de amizades a fim de expandir sua rede de conexões. A idéia desse sistema de recomendação é aprimorar a experiência do usuário na plataforma, de forma a encorajar o crescimento e relevância da rede social para o usuário. Algoritmos como o do *Facebook* são baseados nas relações estruturais entre usuários, formando uma rede de usuários, e não em notas e avaliações, como nos casos da *Amazon.com* e *Netflix* (WOLFRAM, 2013). Com isso, as características desses algoritmos diferem muito dos clássicos algoritmos que tratam do problema de predizer uma nota a partir de uma combinação usuário-item. O leitor que deseja se aprofundar em algoritmos de recomendação baseados em relações estruturais (a exemplo das sugestões de amizade no *Facebook* e classificação de *websites* na página de busca do *Google*) pode ler mais sobre o assunto no Capítulo 10 de (AGGARWAL, 2016).

1.3 Objetivos de sistemas de recomendação

O objetivo final de um sistema de recomendação sempre será a geração de receita. Em um cenário de vendas *online* como na *Amazon*, recomendações corretas impactam diretamente no número de itens vendidos e consequentemente na geração de receita. Para produtos vendidos a partir de assinaturas mensais ou anuais, como *Netflix* e *Spotify*, recomendações corretas melhoram a experiência do usuário, o que em consequência diminui as chances de que o usuário cancele a sua assinatura. Para redes sociais, como *Facebook* e *LinkedIn*, recomendações de novos amigos aumentam o uso e relevância da plataforma no dia a dia dos usuários, alavancando novas formas de receita como publicidade e serviços financeiros.

De maneira geral, problemas de recomendação podem ser formulados de duas maneiras: problemas de previsão e problemas de *ranking*. Problemas de previsão buscam prever a nota dada por um certo usuário a um item específico. De forma mais direta, o objetivo final desses algoritmos é calcular o valor da nota para uma certa combinação usuário-item. Problemas de *ranking*, por sua vez, têm como objetivo gerar uma lista dos k itens de maior interesse para o usuário. Algoritmos de previsão são mais genéricos no sentido de que, uma vez com as notas de cada combinação usuário-item calculadas, basta classificá-las de maneira decrescente para cada usuário e escolher os k primeiros itens (AGGARWAL, 2016).

1.4 Modelos básicos em sistemas de recomendação

Entre as estratégias mais populares para a construção de sistemas de recomendação, duas se destacam: Filtragem colaborativa e Filtragem baseada em conteúdo. Um

sistema de recomendação pode se basear em uma ou outra estratégia, ou mesmo em uma combinação das duas.

Sistemas de recomendação cuja filtragem é baseada em conteúdo se fundamentam na criação de perfis, sendo esses perfis de usuários ou de produtos. Enquanto o perfil de um usuário poderá contar com informações como idade e renda, por exemplo, o perfil de um produto pode levar em conta atributos como marca e ano de lançamento. Um exemplo interessante da aplicação de filtragem baseada em conteúdo na indústria musical é o *"Music Genome Project"* (Projeto genoma da música, na tradução literal), do aplicativo *Pandora*. O projeto conta com um time de musicólogos que é reponsável por coletar centenas de atributos das milhões de músicas disponíveis na plataforma. Considerando o histórico de interesse de cada ouvinte, o sistema é capaz de gerar recomendações personalizadas (KOREN; BELL; VOLINSKY, 2009). Essa estratégia é ilustrada no lado esquerdo da Figura 3.

As aplicações de filtragem colaborativa por sua vez não dependem da criação de perfis de usuários ou produtos, mas levam em conta a relação entre usuários e a interdependência entre produtos para gerar recomendações. Interesses do usuário 1 poderão ser recomendados ao usuário 2 na medida em que os usuários 1 e 2 forem similares, conforme ilustrado no lado direito da Figura 3.

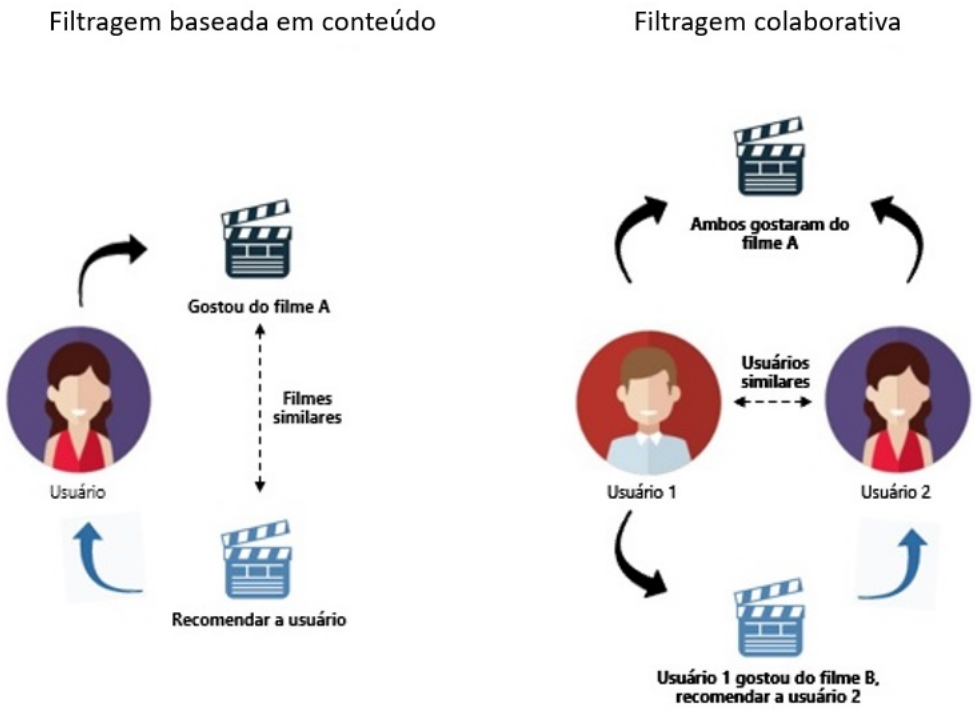

Figura 3 – Filtragem baseada em conteúdo vs. Filtragem colaborativa. Fonte: (TONDJI, 2018)

1.4.1 Sistemas de filtragem colaborativa

Através das opiniões de vários usuários sobre vários itens, os sistemas de filtragem colaborativa computam as recomendações. As notas das avaliações feitas pelos usuários para cada item são dispostas em uma matriz $m \times n$ para m usuários e n itens. Como nem todos os usuários podem comprar ou avaliar todos os n itens, as matrizes utilizadas em algoritmos de filtragem colaborativa tendem a ser esparsas, ou seja, uma grande parte dos elementos da matriz não são especificados. Os elementos especificados da matriz são os elementos para os quais o usuário forneceu alguma informação sobre o item, como por exemplo a nota de uma avaliação.

A Tabela 1 representa uma matriz $m \times n$. Nela estão as notas, obtidas a partir de um intervalo discreto de 1 a 5, que foram dadas pelos usuários aos itens em questão. Os itens não especificados, denotados pelo símbolo de interrogação em vermelho, ?, são os itens não avaliados pelos usuários.

Usuário/Item	Item A	Item B	Item C	...	Item n
Usuário 1	?	?	4	...	?
Usuário 2	1	4	5	...	?
Usuário 3	?	?	?	...	4
Usuário 4	5	2	3	...	4
⋮	⋮	⋮	⋮	⋮	⋮
Usuário m	3	?	?	...	2

Tabela 1 – Tabela com avaliações dadas por m usuários a n itens

O conceito central em algoritmos de filtragem colaborativa é o de que os elementos não especificados podem ser calculados a partir das similaridades entre usuários ou itens. Na grande maioria das vezes, o nível de similaridade entre um grupo de usuários ou um grupo de itens é muito alto, de forma que podemos deduzir a nota que o usuário 1 daria ao item A ao ver a nota que o usuário 2 deu a esse mesmo item, entendendo que os usuários 1 e 2 têm preferências muito parecidas.

Sistemas de filtragem colaborativa podem ser classificados ainda como sistemas baseados em vizinhanças e sistemas baseados em modelos.

1.4.1.1 Sistemas baseados em vizinhanças

Sistemas baseados em vizinhanças podem ser trabalhados sob o ponto de vista da similaridade entre usuários ou itens. No primeiro caso, a vizinhança V é determinada pelo conjunto dos k usuários mais similares ao usuário i em questão. A fim de prever a avaliação que o usuário i daria ao item j, basta calcular a média ponderada das notas dadas pelos usuários da vizinhança V ao item j. Nesse caso, a similaridade é calculada

entre as linhas da matriz $m \times n$. Por outro lado, sob o ponto de vista da similaridade entre itens, a vizinhança V é determinada pelo conjunto dos k itens mais similares ao item j em questão. A lógica segue a mesma: a média ponderada das notas dadas pelo usuário i aos k itens mais similares a j será a nota predita para o item j. Aqui, a similaridade é calculada entre as colunas da matriz $m \times n$.

Por sua simplicidade, sistemas baseados em vizinhanças são populares e de fácil implementação. No entanto, não costumam performar bem em matrizes esparsas, afinal, se existem muitos elementos não especificados o processo de determinar a vizinhança é comprometido, e os resultados podem não ser confiáveis. Uma alternativa que tem sido utilizada para contornar esse problema é a utilização de técnicas de fatoração de matrizes combinadas com algoritmos de otimização, que permitem estimar as avaliações não especificadas com base na estrutura da matriz (KOREN; BELL; VOLINSKY, 2009).

1.4.1.2 Sistemas baseados em modelos

Sistemas baseados em modelos, por sua vez, utilizam técnicas de aprendizado de máquina e mineração de dados. Uma diferença importante entre os algoritmos baseados em vizinhanças e os algoritmos baseados em modelos é a de que, no primeiro caso, os cálculos são realizados para um caso por vez, enquanto no segundo, um modelo generalizado que pode ser aplicado a todo o conjunto de dados é gerado desde o início. Por exemplo, em modelos de vizinhança, selecionamos os k vizinhos mais similares a cada usuário ou item todas as vezes que quisermos calcular uma nova previsão. Em contrapartida, em um sistema baseado na fatoração de matrizes, as avaliações de todos os usuários para todos os itens é feita simultaneamente.

1.4.2 Sistemas de filtragem baseada em conteúdo

Algoritmos de filtragem baseada em conteúdo utilizam as características de itens ou usuários a fim de gerar recomendações. Levando em conta as características dos produtos comprados ou melhor avaliados pelo usuário, o modelo é capaz de prever qual nota um usuário possivelmente daria a um item desconhecido.

Sistemas de filtragem baseada em conteúdo são especialmente úteis em situações onde as avaliações de outros usuários são escassas ou não existem, uma vez que utilizam unicamente dados históricos do próprio usuário para o qual a previsão está sendo gerada. Além disso, diferentemente dos sistemas de filtragem colaborativa, os modelos gerados por algoritmos de filtragem baseada em conteúdo são específicos para cada usuário (AGGARWAL, 2016). Tome, por exemplo, uma matriz de avaliação inicialmente esparsa $m \times n$, com as notas de m usuários para n itens. Ao executarmos um algoritmo de filtragem colaborativa, a matriz completa nos trará a estimativa de todas as possíveis avaliações de todos os usuários para todos os itens. De maneira distinta, para performar uma regressão

ou classificação baseada em conteúdo, consideramos uma matriz cujas linhas são os itens comprados ou avaliados especificamente por um usuário, e cujas colunas são as características desses itens - ou seja, o conteúdo. Dessa forma, os modelos gerados são específicos para o usuário em questão.

1.4.3 Comparação entre modelos colaborativos e baseados em conteúdo

Uma das grandes vantagens de algoritmos baseados em conteúdo é a capacidade de gerar recomendações de novos itens para usuários antigos. Basta compilar as características de um novo item e o modelo utilizará informações históricas sobre o comportamento de compra e avaliação do usuário para predizer se o mesmo irá ou não gostar da novidade. Em contrapartida, esses modelos tendem a gerar recomendações pouco diversas, já que raramente levam em conta itens pouco familiares ao usuário e não utilizam dados provenientes de outros usuários para enriquecer as recomendações.

Quando consideramos a entrada de novos usuários, sistemas de filtragem colaborativa tendem a obter melhor performance, uma vez que novos usuários não têm um histórico robusto de compras e avaliações e portanto não existe insumo para gerar recomendações a esses usuários em um sistema baseado em conteúdo (AGGARWAL, 2016).

1.4.4 Outros modelos em sistemas de recomendação

Apesar dos modelos de filtragem colaborativa e filtragem baseada em conteúdo serem os mais populares no estudo de sistemas de recomendação, existem ainda outros, a exemplo dos sistemas baseados em conhecimento - onde conhecimento prévio e explicitamente fornecido sobre as preferências do usuário é usado com o propósito de gerar recomendações - sistemas demográficos e sistemas utilitários. O leitor interessado pode encontrar maiores detalhes sobre esses outros modelos a partir da seção 1.3.3 do primeiro capítulo em (AGGARWAL, 2016).

1.5 Como avaliar sistemas de recomendação

Como mencionado anteriormente, o objetivo final de qualquer sistema de recomendação é gerar receita. Dessa forma, a melhor maneira de avaliar a efetividade de um sistema de recomendação é entender se os itens recomendados são eventualmente consumidos. Com esse fim são implementados os populares "testes A/B", onde um grupo de usuários é selecionado e então dividido em dois subgrupos A e B, cada um sendo exposto a um sistema de recomendação diferente e mantendo todas as outras variáveis constantes. Os subgrupos são então comparados com relação a uma ou mais métricas, como por exemplo a taxa de conversão (um item sugerido que é consumido dentro de

um período pré-especificado conta uma conversão). O grupo com melhor performance representa o melhor sistema de recomendação.

No entanto, realizar testes A/B requer acesso a uma rede *online* que possibilita que um sistema seja implementado e em seguida seus resultados monitorados. Como obter acesso a uma rede *online* não é tarefa trivial, sendo uma oportunidade reservada apenas àquelas instituições que fornecem os produtos consumidos pelo usuário final, foram desenvolvidos também testes *offline*, amplamente utilizados tanto em pesquisa como comercialmente. De maneira prática, testes *offline* auxiliam na pré-seleção de sistemas de recomendação que serão implementados de maneira *online* (GEBREMESKEL; VRIES, 2016).

Os algoritmos de sistemas de recomendação implementados no Capítulo 3 desse trabalho serão testados de maneira *offline*. São discutidas a seguir algumas entre as métricas mais populares utilizadas em testes *offline* para a avaliação de sistemas de recomendação.

1.5.1 Precisão

A precisão de um sistema de recomendação pode ser entendida de duas maneiras. Para sistemas cuja função é predizer as notas das avaliações de uma combinação usuário-item, a precisão será a medidade de quão diferente é a nota predita da nota real. Em contrapartida, para sistemas cuja função é criar um *ranking* com os k itens de maior interesse para o usuário, a precisão é a medida de quantos itens dentre esses k foram recomendados corretamente (AGGARWAL, 2016).

1.5.1.1 Precisão em problemas de previsão

Seja D um conjunto de dados totalmente conhecido referentes a m usuários e n itens. Sejam r_{ij} a nota real dada pelo usuário i ao item j, e $\hat{r}_{ij}$ a nota dada pelo usuário i ao item j predita pelo algoritmo. A diferença entre a nota real e a nota predita será o erro para essa combinação usuário-item. Podemos calcular o erro médio para todo o conjunto de dados D através da fórmula 1.1 abaixo.

$$MAE = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}|\hat{r}_{ij} - r_{ij}|}{mn} \tag{1.1}$$

Onde MAE é o erro absoluto médio (do inglês *mean average error*). O valor do MAE será o valor médio de diferença entre a nota prevista e a nota real para cada combinação usuário-item no conjunto de dados (KANE, 2018). É possível também encontrar o NMAE, que nada mais é do que o MAE normalizado, conforme equação 1.2 (AGGARWAL,

2016). Como o valor do MAE normalizado vai de 0 a 1, o resultado do NMAE é talvez mais intuitivo, sendo que quanto mais próximo de zero melhor.

$$NMAE = \frac{MAE}{r_{max} - r_{min}} \tag{1.2}$$

Onde r_{max} e r_{min} são a maior e menor nota no conjunto de dados D, respectivamente.

Dependendo da aplicação, grandes discrepâncias entre o valor predito e o valor real são especialmente prejudiciais ao sistema. Por exemplo, imagine uma aplicação na indústria farmacêutica, onde pequenas diferenças entre os volumes das substâncias podem resultar em cenários muito diferentes. Nesses casos, é possível utilizar o erro quadrático médio (ou apenas MSE, do inglês *mean squared error*), que por conta do fator quadrático penaliza de maneira mais acentuada as maiores diferenças entre as predições e os valores reais (AGGARWAL, 2016). Calcula-se o MSE conforme equação 1.3 abaixo.

$$MSE = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{r}_{ij} - r_{ij})^2}{mn} \tag{1.3}$$

Através do MSE é possível também encontrar o RMSE, que nada mais é do que a raiz quadrada do MSE. A vantagem de se utilizar o RMSE ao invés do MSE é que o valor do RMSE tem a mesma unidade de medida que as notas dadas pelos usuários. Abaixo são apresentadas as equações para o RMSE e o NRMSE, sendo o último o RMSE normalizado (KANE, 2018).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{r}_{ij} - r_{ij})^2}{mn}} \tag{1.4}$$

$$NRMSE = \frac{RMSE}{r_{max} - r_{min}} \tag{1.5}$$

É importante lembrar que todas as métricas de precisão discutidas acima são métricas que calculam o erro entre a predição e o valor real, e devem ser portanto minimizadas pelo sistema.

1.5.1.2 Precisão em problemas de *ranking*

Problemas de *ranking* têm como objetivo criar uma lista de k itens a serem recomendados para o usuário. A fim de calcular a precisão de um sistema de recomendação baseado em *ranking*, podemos calcular quantas das recomendações feitas pelo sistema foram realmente avaliadas pelo usuário. O resultado desse cálculo será o que chamados de taxa de acerto (KANE, 2018).

Por exemplo, suponha que o algoritmo retorne três sugestões de filmes para o usuário: "Gladiador", "Os Incríveis" e "Como se fosse a primeira vez". Desses três filmes, o usuário avaliou apenas "Os Incríveis". Nesse caso, consideramos apenas um acerto. A fim de encontrar a taxa de acerto para todo o sistema, somamos todos os acertos individuais para cada usuário e dividimos pelo número total de usuários (KANE, 2018).

De maneira geral, seja R_i o conjunto de k itens recomendados pelo algoritmo para um usuário i fixado, e A_i o conjunto de todos os itens que esse mesmo usuário já avaliou. A taxa de acerto do sistema pode ser calculada conforme 1.6 abaixo.

$$\text{Taxa de acerto} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{k} C_{ij}}{m} \tag{1.6}$$

Onde m é o número total de usuários, $C_{ij} = 1$ caso o item $j \in R_i$ esteja contido também em A_i, e $C_{ij} = 0$ caso contrário.

Uma outra forma de avaliar a precisão de um sistema de recomendação com foco em *ranking* é calcular a correlação entre a posição relativa de um item na lista dos k itens recomendados pelo sistema a um usuário i e a posição relativa desse mesmo item na lista dos k itens mais bem avaliados por esse usuário. Essa correlação pode ser calculada de diversas maneiras, e entre as mais comuns está o coeficiente de correlação de postos *Spearman*, conforme equação 1.7 abaixo.

$$\rho = \frac{\sum_{j=1}^{k}(r_j - \bar{r})(\hat{r}_j - \bar{\hat{r}})}{\sqrt{\sum_{j=1}^{k}(r_j - \bar{r})^2 \sum_{j=1}^{k}(\hat{r}_j - \bar{\hat{r}})^2}} \tag{1.7}$$

O valor de ρ na equação 1.7 acima varia entre -1 e 1. Quando $\rho = 1$ ou $\rho = -1$, temos uma relação monotônica entre as posições relativas em cada uma das listas, sendo que $\rho = 1$ representa uma função estritamente crescente (quanto maior a posição relativa na lista de itens recomendados, maior a posição relativa na lista dos itens mais bem

avaliados) e $\rho = -1$ representa uma função estritamente decrescente. Quando $\rho = 0$ não existe correlação monotônica entre as duas listas.

A correlação de *Spearman* média entre todos os m usuários pode ser calculada a fim de se ter uma métrica única para todo o sistema.

1.5.2 Cobertura

A cobertura é o percentual de itens que podem ser recomendados através do algoritmo em comparação com o número total de itens existentes. Por exemplo, quando um livro novo é lançado na plataforma de comércio eletrônico *Amazon.com*, ele só aparecerá como uma recomendação após ser vendido algumas vezes. Isso acontece porque o algoritmo precisa dessas vendas para presumir algum padrão de compra e avaliação do item. Até que esse padrão seja estabelecido, existirão itens disponíveis na plataforma que não serão sugeridos pelo sistema de recomendação, diminuindo a métrica de cobertura (KANE, 2018).

Considere novamente o conjunto D de dados conhecidos referentes a m usuários e n itens. Seja T_i a lista dos k itens recomendados ao usuário i pelo sistema, sendo $i = 1, ..., m$. Podemos calcular a cobertura do sistema através da equação 1.8 (AGGARWAL, 2016) (LI, 2019).

$$Cobertura = \frac{|| \cup_{i=1}^{m} T_i ||}{n} \tag{1.8}$$

Onde n é o número total de itens disponíveis para consumo.

1.5.3 Diversidade

Bons sistemas de recomendação apresentam ao usuário recomendações relevantes porém diversas. Recomendar sempre o mesmo produto ou produtos muito similares pode sinalizar um sistema de recomendação relevante, mas pouco diverso. Em contrapartida, recomendar uma lista de produtos aleatórios é extremamente diverso, porém pode ser pouco relevante. A diversidade pode ser calculada através de uma métrica de similaridade para cada par de itens. Se a lista de produtos recomendada ao usuário é muito similar entre si, a diversidade é baixa. Do contrário, a diversidade é alta (KANE, 2018).

Podemos calcular a similaridade a cada par de itens em uma lista de k itens recomendados pelo sistema através da similaridade de cossenos. Para isso, no entanto, é necessário termos informações sobre as características dos itens recomendados. Sejam por exemplo os itens A, B e C três produtos disponíveis para compra e recomendados a um usuário i qualquer. Exemplificamos abaixo com uma tabela os detalhes das características de cada um dos produtos.

Produto	Categoria	Faixa de preço	Marca
A	Infantil	Até R$ 25	PomPom
B	Casa	De R$ 25 a R$ 50	Comfort
C	Infantil	De R$ 25 e R$ 50	Comfort

Tabela 2 – Detalhamento das características dos produtos A, B e C

A Tabela 2 acima pode ser transformada em uma tabela binária, conforme Tabela 3.

Produto	Infantil	Casa	Até R$ 25	De R$ 25 a R$ 50	PomPom	Comfort
A	1	0	1	0	1	0
B	0	1	0	1	0	1
C	1	0	0	1	0	1

Tabela 3 – Tabela binária gerada a partir do detalhamento das características dos produtos A, B e C

Considerando cada linha da Tabela 3 como um vetor, é possível encontrar o cosseno do ângulo entre os vetores **A** e **B**, **B** e **C** e **A** e **C**. Vetores muito similares apontam para direções similares, e o ângulo entre esses vetores se aproxima de zero. Como o cosseno de zero é 1, quanto mais próximo de 1 é o cosseno do ângulo entre dois vetores, mais similares são esses vetores. Para encontrar uma métrica única de similaridade S para essa lista de recomendações, podemos somar os valores de cada cosseno e dividir pelo número total de itens (três valores de cosseno e três itens nesse exemplo). A métrica de diversidade será igual a $D = 1 - S$ (KANE, 2018). A fim de encontrar uma métrica de diversidade para todo o sistema, basta somar cada D_i, sendo $i = 1, ..., m$, e dividir por m, o número total de usuários.

1.5.4 Novidade

Novidade é a métrica de quão populares são os itens recomendados. Usuários tendem a confiar mais em recomendações que incluem pelo menos alguns itens familiares, afinal, itens populares tendem a ser populares exatamente porque são desejados por grande parte do público. No entanto, ao recomendar apenas itens que estão sempre entre os mais vendidos, o sistema falha em sua missão de apresentar ao usuário produtos que talvez ele não conheça, mas que podem ser muito relevantes. É importante lembrar que o objetivo final de um sistema de recomendação sempre é gerar receita, e expor o usuário a mais itens relevantes aumentam as chances de compra. Quanto maior o número de itens populares em uma lista de recomendações, menor a novidade (KANE, 2018).

Em casos onde existe o acesso a um sistema *online*, a métrica de novidade pode ser medida através de uma pesquisa onde os usuários respondem se já conheciam ou

não os itens recomendados pelo sistema. De maneira *offline* é possível estimar um valor para novidade caso o conjunto de dados com as notas para cada combinação usuário-item possua também a informação de data e hora em que a avaliação foi feita (AGGARWAL, 2016). A idéia principal é a de que um sistema que recomenda itens desconhecidos ao usuário incentiva ações futuras ao invés de ações no presente. Com isso em mente, é determinada uma data de corte t_0, tal que todas as avaliações feitas após a data de corte são removidas do conjunto de dados. Além disso, algumas recomendações anteriores à t_0 também são removidas. Feito isso, o conjunto de dados restante é utilizado para construir o algoritmo de recomendação. Após a construção, o algoritmo é executado e os resultados analisados. Para cada item recomendado pelo algoritmo e que foi avaliado antes de t_0 a métrica de novidade é penalizada, e para cada item recomendado e avaliado após t_0 a métrica de novidade é premiada. A idéia aqui é a de que, se o sistema recomendou itens que já haviam sido avaliados então as recomendações foram óbvias, no sentido de que não trouxeram algo novo para o usuário. Do contrário, a métrica de novidade deve aumentar.

2 Modelos matemáticos

2.1 Introdução ao aprendizado de máquina

Antes de entrarmos no detalhamento matemático dos algoritmos utilizados em sistemas de recomendação, faremos uma breve introdução sobre as terminologias e conceitos de aprendizado de máquina que serão utilizados neste trabalho. Uma referência mais completa sobre o tema pode ser encontrada em (KUBAT, 2017) e (BISHOP, 2006).

2.1.0.1 Terminologia e estrutura de dados

Esse trabalho ficará restrito ao estudo de dados tabulares estruturados. Nesse caso, assumiremos que os dados necessários para a resolução dos problemas de recomendação podem ser expressos na forma de uma matriz com m linhas e n colunas, cujas informações podem conter dados numéricos, categóricos ou textuais. Assim sendo, por delimitação de escopo, não trataremos de dados no formato de imagens, nem mesmo dados não estruturados.

De maneira geral, um algoritmo de aprendizado de máquina recebe um conjunto de informações - ao que chamamos atributos de entrada - e nos retorna um conjunto de respostas, também conhecidas como atributos de saída. Os atributos de entrada são fornecidos ao algoritmo no formato de matrizes $m \times n$, onde cada linha da matriz representa uma observação, também chamada de instância, e possui n colunas. Podemos definir cada observação como um vetor $\mathbf{x_i}$, $i = 1, ..., m$, e cada um dos n elementos do vetor $\mathbf{x_i}$ dizem respeito a n variáveis preditivas. O atributo de saída, por sua vez, será um vetor $\hat{\mathbf{y}} \in R^m$. Cada elemento $\hat{y_j}$ do vetor $\hat{\mathbf{y}}$ é uma resposta do algoritmo à observação j da matriz $X_{m \times n}$ de atributos de entrada (veja a Figura 4). Idealmente os atributos de entrada devem ser independentes entre si, enquanto o atributo de saída será uma variável dependente, sendo calculada a partir das variáveis preditivas.

No vetor $\hat{\mathbf{y}}$ são armazenadas as previsões dos valores dos atributos de saída feitas pelo algoritmo. Em contrapartida, o vetor atributo alvo $\mathbf{y}, \mathbf{y} \in R^m$, contém as informações reais e previamente conhecidas. Essas informações são utilizadas na construção do modelo e são o insumo do aprendizado do algoritmo. Na prática, queremos aperfeiçoar esse aprendizado a fim de que $\hat{\mathbf{y}}$ se aproxime ao máximo de $\mathbf{y}$. Chamamos de erro à função que calcula as imprecisões do modelo se baseando nas diferenças entre os vetores $\hat{\mathbf{y}}$ e $\mathbf{y}$ (MARSLAND, 2009).

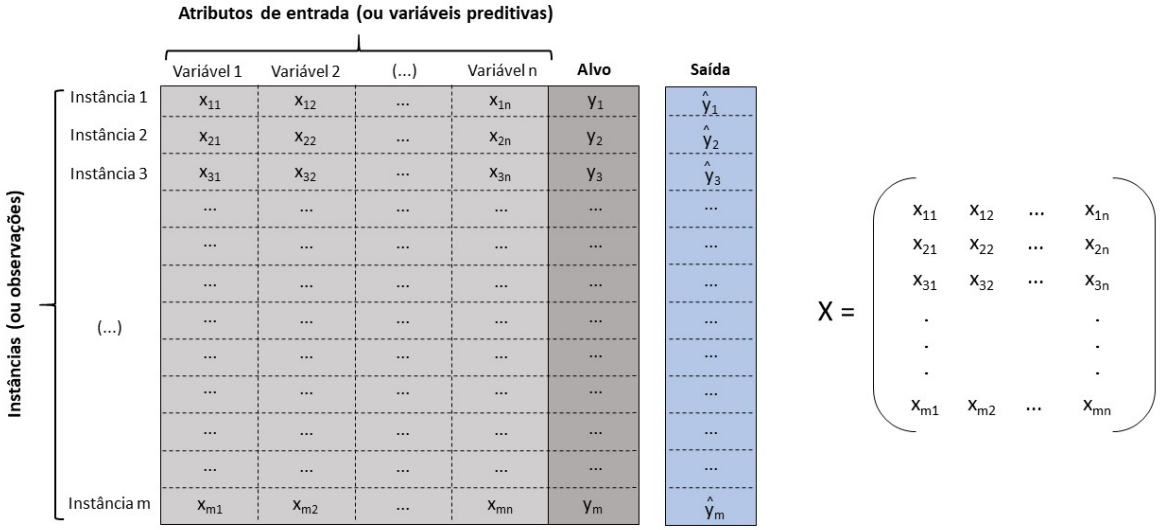

Figura 4 – Desenho esquemático de uma base de dados e seus elementos para um algoritmo de aprendizado de máquina. Fonte: autora.

De forma a aplicar os conceitos discutidos acima em um contexto de sistemas de recomendação, imagine uma base de dados com n variáveis preditivas, nos moldes da base de dados apresentada na Figura 4, sendo cada variável uma característica musical. A primeira variável pode, por exemplo, ser o gênero da música, sendo as possíveis opções algo como rock, samba, sertanejo e etc. São apresentadas em seguida as demais variáveis, que poderiam ser algo como duração da música e ano de lançamento, entre outras. Suponha que a base tenha m instâncias, cada uma sendo uma música diferente, e que a variável dependente seja um valor binário, de modo que o valor 1 indica o equivalente a uma "curtida" - ou seja, um feedback positivo a partir do usuário para aquela música - e 0 a falta de interação entre o usuário e a música. Um desenho esquemático é fornecido na Figura 5.

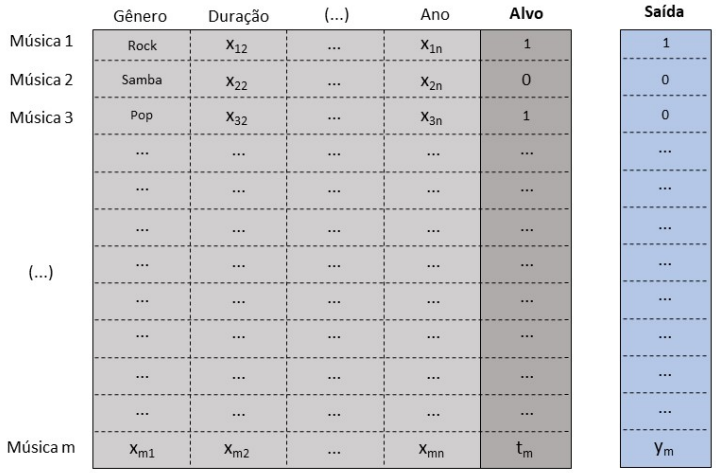

Figura 5 – Desenho esquemático de uma base de dados referente a m músicas e n características. Fonte: autora.

Veja que os vetores $\mathbf{y}$ e $\hat{\mathbf{y}}$ referentes ao atributo alvo e ao atributo de saída não são completamente iguais. Isso é esperado, uma vez que na maioria dos casos o algoritmo não é capaz de reproduzir os valores do atributo alvo com 100% de precisão. O objetivo do algoritmo é justamente aprender com os dados previamente conhecidos, de modo a errar cada vez menos em suas previsões, representadas no vetor $\hat{\mathbf{y}}$.

2.1.0.2 Aprendizado supervisionado e não supervisionado

Os métodos de aprendizado de máquina podem ser classificados em duas principais categorias: o aprendizado supervisionado e o não supervisionado. No aprendizado supervisionado é necessário fornecer ao algoritmo o que chamamos de "verdade fundamental", ou seja, o algoritmo deverá receber algum conhecimento prévio sobre o valor e o tipo de atributo de saída esperado. De modo geral, a tarefa do algoritmo é estabelecer uma correspondência entre os atributos de entrada e os atributos de saída. Por outro lado, para os algoritmos de aprendizado não supervisionado, não é necessário fornecer qualquer tipo de conhecimento prévio sobre o atributo de saída. Nesses casos, o objetivo do algoritmo é descobrir as estruturas implícitas ao conjunto de dados em questão (SONI, 2018).

De forma a ilustrar os conceitos de aprendizado supervisionado e não supervisionado, imagine ensinar a uma criança sobre cores. Ao mostrá-la um objeto azul, você diz a ela "essa cor é a cor azul", e da mesma maneira com a cor rosa, vermelha, verde, e assim por diante. Isso é um aprendizado supervisionado, pois as características do atributo de

saída são especificadas previamente. Em contrapartida, aprender a falar é um exemplo de aprendizado não supervisionado. Nesse caso, a própria criança começa a entender a estrutura de linguagem ao ouvir outras pessoas falarem. Mais tarde, em complemento, a criança pode ter contato com um método supervisionado para aperfeiçoar a fala através do aprendizado de regras gramaticais.

Algoritmos de aprendizado supervisionado são geralmente construídos como algoritmos de regressão ou classificação. Algoritmos de regressão tem como objetivo mapear atributos de entrada a um atributo de saída numérico. Por exemplo, dado um conjunto de características de clientes, é possível determinar a probabilidade de certo cliente comprar um certo produto através de um algoritmo de regressão. Um caso específico de algoritmos de regressão são os algoritmos de classificação, onde o atributo de saída assume um formato categórico. Pode-se, por exemplo, utilizar um algoritmo de classificação para recomendar a um cliente um próximo destino de férias a partir de suas últimas viagens.

As tarefas mais comuns em aprendizado não supervisionado são clusterização e redução de dimensionalidade. Em clusterização é possível encontrar tendências em dados de difícil interpretação humana através da criação de clusters, ou seja, grupos de dados que possuem características semelhantes. Similarmente, a redução de dimensionalidade é utilizada a fim de encontrar quais os fatores latentes em um grupo de dados representado por uma matriz esparsa, possibilitando a representação dessa matriz de muitas dimensões através de sua decomposição em matrizes de dimensões menores (SONI, 2018) .

2.1.0.3 Dados de treinamento, validação e teste

Durante a construção de um algoritmo de aprendizado de máquina, é necessário utilizar três conjuntos de dados diferentes: dados de treinamento, dados de validação e dados de teste. Os dados de treinamento serão efetivamente utilizados pelo algoritmo para o aprendizado, enquanto os dados de validação e teste ficarão ocultos do aprendizado e entrarão em uma fase posterior apenas para verificar a precisão do modelo. Os dados de validação são utilizados para fazer o ajuste fino dos parâmetros do algoritmo com o objetivo de melhorar sua acurácia, enquanto os dados de teste devem ser usados apenas no modelo final, completamente especificado, de forma a antecipar o resultado do algoritmo sobre dados reais e desconhecidos (ACADEMY, 2021).

De volta ao exemplo da base de dados de características musicais, suponha que a base de dados com as características musicais previamente conhecidas possua mil observações, ou seja, $m = 1.000$. Dispomos então de mil casos onde temos todas as características da música, e ao final um valor binário representando o feedback do usuário para aquela música. A fim de construir o algoritmo, separamos uma fração dessas mil observações para serem nossos dados de treinamento. Como sugestão, pode-se utilizar cerca de 70% das observações como dados de treinamento, 5% como validação e 25% como

teste. No entanto, esses percentuais podem variar em função do número total de instâncias e da natureza do problema.

É importante que sejam utilizados três conjuntos de dados completamente diferentes e construídos de maneira aleatória a fim de minimizar o viés do modelo. Caso seja feita a construção, validação e teste do algoritmo com o mesmo conjunto de dados, durante a fase de teste o algoritmo pode apresentar um desempenho muito bom, o que pode não se confirmar quando for exposto a um conjunto de dados desconhecidos. Essa falsa impressão se origina do fato de que estamos testando o algoritmo com os mesmos dados que utilizamos para ensiná-lo, ou seja, o algoritmo tem tudo "decorado". Esse fenômeno, onde o algoritmo "decora" os dados, é chamado de *overfitting*, e é algo que queremos evitar ao construir um modelo. Se *overfitting* ocorre o modelo será incapaz de generalizar as regras, o que aumenta o risco de que sejam geradas previsões incorretas (AGGARWAL, 2016).

2.2 Introdução à algoritmos para sistemas de recomendação

Como discutido no primeiro capítulo, existem de maneira geral duas categorias em sistemas de recomendação: sistemas cuja filtragem é baseada em conteúdo e sistemas de filtragem colaborativa. Sistemas de filtragem colaborativa são subdivididos em dois grupos, os sistemas baseados em vizinhança e os baseados em modelos.

Esse trabalho abordará cinco algoritmos de aprendizado de máquina no contexto de sistemas de recomendação, sendo eles a regressão linear, o método dos k-vizinhos mais próximos, a redução de dimensionalidade, as árvores de decisão e as redes neurais artificiais. Alguns algoritmos são comumente utilizados em problemas de recomendação de ambas as categorias, enquanto outros, por sua natureza e limitações técnicas, são usados especificamente em problemas de apenas uma categoria. No Capítulo 3 é realizada a aplicação dos algoritmos dos k-vizinhos mais próximos, redução de dimensionalidade de matrizes esparsas e redes neurais artificiais.

Na Figura 6 abaixo são esquematizadas as subdivisões de sistemas de recomendação e os algoritmos que podem ser utilizados em problemas de cada uma dessas subdivisões.

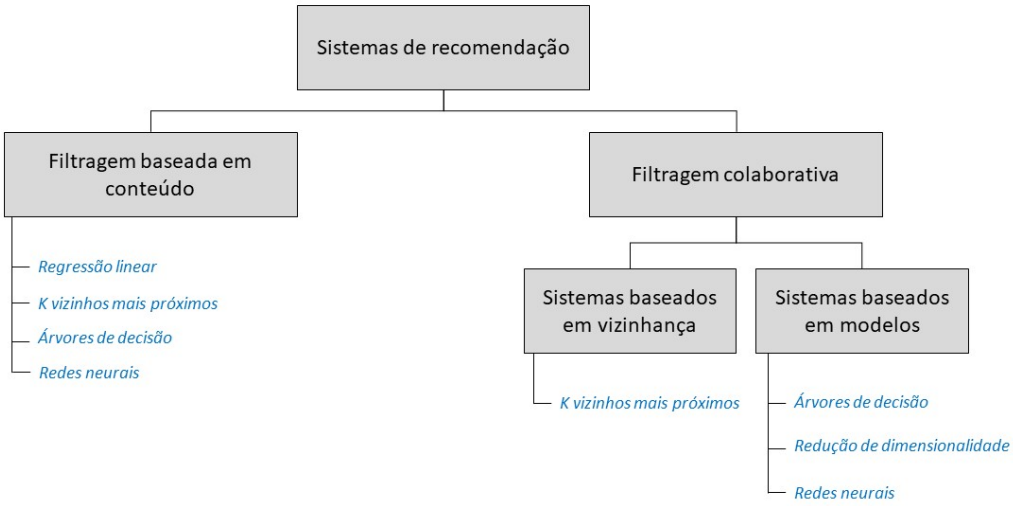

Figura 6 – Resumo das várias categorias de sistemas de recomendação com seus respectivos algoritmos. Fonte: autora.

Nas seções a seguir serão apresentados os algoritmos citados na Figura 6, procurando enfatizar os aspectos matemáticos por trás de cada um, assim como exemplos no contexto de sistemas de recomendação e os respectivos algoritmos de implementação.

2.3 Regressão Linear

O objetivo da regressão linear é encontrar uma equação linear que explique a relação entre as variáveis de previsão - ou atributos de entrada - e a variável de resposta, a qual também chamamos de atributo de saída.

2.3.1 Embasamento matemático

Neste método, assume-se a hipótese de que a variável alvo, $\mathbf{y} \in R^m$, pode ser explicada por meio de uma relação linear com as variáveis de previsão. Definiremos aqui cada variável de previsão como $\mathbf{x_j} \in R^m$, onde $j = 1, 2, ..., n$. A relação entre a variável alvo $\mathbf{y}$ e as variáveis de previsão $\mathbf{x_j}$ pode ser explicada através da equação 2.1:

$$\mathbf{y} = c_0 + c_1\mathbf{x_1} + c_2\mathbf{x_2} + ... + c_j\mathbf{x_j} + ... + c_n\mathbf{x_n} \tag{2.1}$$

Nesse contexto, um problema de regressão linear consiste em determinar os coeficientes $\{c_0, c_1, ..., c_n\}$ que permitem encontrar estimativas que sejam as mais próxi-

mas possíveis dos dados conhecidos. O objetivo do modelo é "aprender" com os dados previamente conhecidos quais são os coeficientes mais adequados para aquele problema.

Do ponto de vista matemático, cada instância do conjunto de treinamento fornecerá uma equação linear cujas variáveis são os coeficientes $\{c_0, c_1, ..., c_n\}$. O conjunto dessas equações forma um sistema linear, que pode ser escrito na forma matricial conforme equação 2.2.

$$X\mathbf{c} = \mathbf{y} \tag{2.2}$$

Onde:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ 1 & x_{31} & x_{32} & \dots & x_{3n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

$$\mathbf{c} = \{c_0, c_1, ..., c_n\} \tag{2.3}$$

$$\mathbf{y} = \{y_1, y_2, ..., y_n\} \tag{2.4}$$

Se $m < n$, ou seja, o número de instâncias do conjunto de treinamento for menor do que o número de variáveis preditivas e consequentemente menor do que o número de coeficientes, o sistema é subdeterminado e acontece o que se chama de hiper-parametrização da regressão, quando o número de variáveis é maior do que o número de dados.

Se $m = n$, o sistema pode admitir uma solução única que, em geral, não serve para o propósito de fazer previsões ou recomendações, visto que há uma interpolação dos pontos. Este tipo de modelo explica perfeitamente os dados de treinamento, mas falha na extrapolação dos resultados. Nesse caso ocorre o *overfitting*, conforme comentado na seção 2.1.

O caso interessante e mais desejável em modelos de aprendizagem ocorre quando $m > n$, ou seja, o número de instâncias é maior (e preferencialmente bem maior) do que o número de variáveis. Nesse caso o sistema linear é sobredeterminado e, em geral, não admite solução. O que se busca então é um conjunto de coeficientes que melhor se aproxime dos dados reais do treinamento.

Formalmente, busca-se encontrar coeficientes que minimizem o erro entre as previsões feitas pelo modelo (armazenadas no vetor $\hat{\mathbf{y}}$) e os valores alvo observado (armazenados no vetor $\mathbf{y}$). Isso é resolvido através dos métodos do quadrados mínimos.

A fim de construir um modelo de regressão linear utilizando o método dos quadrados mínimos, iniciamos pela multiplicação das matrizes na equação 2.2, obtendo o sistema abaixo:

$$\begin{cases} c_0 + x_{11}c_1 + x_{12}c_2 + \cdots + x_{1n}c_n = y_1 \\ c_0 + x_{21}c_1 + x_{22}c_2 + \cdots + x_{2n}c_n = y_2 \\ \vdots \\ c_0 + x_{m1}c_1 + x_{m2}c_2 + \cdots + x_{mn}c_n = y_m \end{cases} \tag{2.5}$$

Em um mundo perfeito, onde todas as variáveis de X que influenciam o resultado de $\mathbf{y}$ são conhecidas, o sistema 2.5 apresentaria uma solução. Como esse não é o caso, o sistema não apresenta solução e portanto não existe um vetor $\mathbf{c}$ que multiplicado à matriz X retorne exatamente os valores em $\mathbf{y}$.

No entanto, é possível encontrar um vetor $\mathbf{p}$ que, multiplicado à matriz X, traga resultados muito próximos aos valores reais de $\mathbf{y}$.

Chamemos de $\hat{\mathbf{y}}$ o vetor proveniente da multiplicação de X e $\mathbf{p}$. O objetivo aqui é encontrar o vetor $\mathbf{p}$ que minimize o erro ϵ, ou seja, que minimize a diferença entre os valores reais em $\mathbf{y}$ e os valores calculados em $\hat{\mathbf{y}}$:

$$\epsilon = ||\mathbf{y} - X\mathbf{p}|| = ||\mathbf{y} - \hat{\mathbf{y}}|| \tag{2.6}$$

Como a diferença entre $\mathbf{y}$ e $\hat{\mathbf{y}}$ nada mais é do que o vetor gerado por $\mathbf{y}$ - $\hat{\mathbf{y}}$ (o qual iremos chamar aqui de $\mathbf{e}$), queremos minimizar a magnitude de $\mathbf{e}$. Sabemos que uma vez que $\hat{\mathbf{y}}$ é a multiplicação entre X e $\mathbf{p}$, o vetor $\hat{\mathbf{y}}$ mais próximo que podemos encontrar de $\mathbf{y}$ é a projeção de $\mathbf{y}$ no subespaço gerado pelos vetores-coluna da matrix X (MARGALIT, 2019), então o erro mínimo será:

$$min\|\mathbf{e}\| = \|\mathbf{y} - proj_{C(X)}\mathbf{y}\| \tag{2.7}$$

Por definição, o vetor gerado por $\mathbf{y} - proj_{C(X)}\mathbf{y}$ é ortogonal ao subespaço gerado pelos vetores-coluna da matrix X, e por consequência pertence ao núcleo de X transposta, X^T. Com isso temos que:

$$\begin{aligned} X^T(\mathbf{y} - proj_{C(X)}\mathbf{y}) &= \mathbf{0} \\ X^T(\mathbf{y} - X\mathbf{p}) &= \mathbf{0} \\ X^T\mathbf{y} - X^TX\mathbf{p} &= \mathbf{0} \end{aligned} \tag{2.8}$$

A equação 2.8 é um sistema linear, e portanto pode-se encontrar o vetor $\mathbf{p}$ que irá minimizar o erro ϵ entre os valores reais do vetor de resposta $\mathbf{y}$ e os valores calculados $\hat{\mathbf{y}}$, uma vez que $X^T\mathbf{y}$ e X^TX são conhecidos. Com isso é possível encontrar a equação linear que correlaciona os valores de X com a variável dependente $\hat{\mathbf{y}}$, de acordo com a equação 2.9 abaixo:

$$\hat{y}_j = p_0 + p_1 x_{j1} + p_2 x_{j2} + \cdots + p_n x_{jn} \qquad \forall j \in 1, ..., m \tag{2.9}$$

2.3.2 Aplicação em sistemas de recomendação

Algoritmos de regressão linear podem ser implementados na construção de sistemas de filtragem baseada em conteúdo, onde a matriz $X_{m \times n}$ compreende n características de m itens para um usuário u fixo. Cada matriz X corresponderá a um usuário único. Abaixo, exemplificamos a aplicação desse algoritmo.

Exemplo 1. ***Regressão linear em sistemas baseados em conteúdo***
Considere uma base de dados gerada a partir das avaliações dadas por um usuário a uma coleção de filmes, conforme a Tabela 4 abaixo.

Código do Filme	Gênero	Nacionalidade	Avaliação
01	Comédia	Americano	3,5
02	Drama	Alemão	2,7
03	Comédia	Brasileiro	3,8
04	Drama	Brasileiro	4,1

Tabela 4 – Avaliações de filmes: Exemplo 1

Para cada filme, representado por um código na primeira coluna da tabela, são fornecidas as informações de gênero e nacionalidade, e por fim a avaliação dada pelo usuário, que pode ir de 1 a 5 estrelas. O objetivo da regressão linear, nesse caso, seria encontrar uma equação linear que relacione as características dos filmes com a avaliação dada pelo usuário. Uma vez com essa regra estabelecida, seria possível em teoria predizer as avaliações que o usuário daria a filmes que ele ainda não viu, tendo insumos para recomendá-los ou não.

A primeira coluna, com o código do filme, é apenas uma coluna de indexação usada para referenciar cada filme. As colunas 2 e 3 são as variáveis de previsão, utilizadas na composição da matriz de variáveis de previsão X. A última coluna, com as avaliações dadas pelo usuário, é o atributo alvo $\mathbf{y}$.

Como os valores das colunas 2 e 3 da tabela acima são categóricos, é necessário primeiro codificá-los a fim de seguir com a regressão linear. Tomemos o conjunto gerado

pelas categorias de gênero $G = \{Comédia, Drama\}$ e o conjunto gerado pelas categorias de nacionalidade $N = \{Americano, Alemão, Brasileiro\}$. Cria-se uma coluna para cada categoria. Caso o filme apresente a característica em questão, o elemento a_{ij} pertencente à matriz de variáveis de previsão X, onde i faz referência ao filme e j à categoria, será igual a 1. Caso contrário, a_{ij} será igual a 0. Esse conceito é conhecido como "one hot encoding", e é exemplificado na Tabela 5 abaixo.

Código	Comédia	Drama	Americano	Alemão	Brasileiro	Avaliação
01	1	0	1	0	0	3,5
02	0	1	0	1	0	2,7
03	1	0	0	0	1	3,8
04	0	1	0	0	1	4,1

Tabela 5 – Variáveis categóricas codificadas

Uma vez com as variáveis categóricas codificadas, a matriz de variáveis de previsão X e o vetor do atributo alvo $\mathbf{y}$ seriam:

$$X = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 3,5 \\ 2,7 \\ 3,8 \\ 4,1 \end{bmatrix}$$

É importante mencionar que, apesar de termos incluido colunas para todas as categorias de gênero e nacionalidade em um primeiro momento, devemos retirar uma coluna de cada variável antes de seguir com a regressão. Isso acontece porque a fim de construir uma regressão linear as variáveis de previsão devem ser independentes entre si. Ao considerar o caso da variável de gênero, por exemplo, sabe-se que, após a codificação, a coluna "Comédia" somada à coluna "Drama" sempre será igual a 1, ou seja, é possível inferir o valor da coluna "Drama" sabendo o valor da coluna "Comédia" e vice versa. Isso significa que as colunas na verdade são combinações lineares e não são independentes entre si. O mesmo acontece no caso da nacionalidade. Ao se retirar uma das colunas, ainda é possível inferir qual o valor da coluna faltante através dos valores das colunas que permanecem (uma vez que a coluna retirada é uma combinação linear das colunas que ficam), o que nos permite seguir com a regressão sabendo que todas as variáveis são independentes entre si. Para esse exemplo, serão retiradas as colunas referentes às variáveis de "Drama" e "Brasileiro". A nova matriz X será:

$$X = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Uma vez com o processo acima finalizado, temos uma matriz X composta apenas por números. O próximo passo é normalizar os dados da matriz. Imagine por exemplo que a matriz em questão tivesse uma coluna com a idade do filme e outra com o valor total, em reais, provenientes dos ingressos vendidos nas bilheterias de cinema durante as semanas de estréia. Enquanto a coluna de idade não passará de talvez 50 ou 60 anos, a coluna com o faturamento dos ingressos poderá chegar aos milhões de reais. Se construimos o modelo sem antes normalizar esses dados, a variável de faturamento terá maior influência no resultado final da regressão, apesar de não necessariamente ser a variável mais importante quando comparada à variável de idade. Por isso, antes de seguir para a etapa de construção do modelo, é importante normalizar os dados em cada coluna de X (LAKSHMANAN, 2019).

Após essa fase de pré-processamento de dados, basta calcular o vetor $\mathbf{p}$ utilizando a equação 2.8. Com os valores dos elementos de $\mathbf{p}$ conhecidos, o valor de $\hat{y}$ referente a uma nova observação m_{j+1} pode ser encontrado a partir da equação 2.9.

No Algoritmo 2.1 apresentamos uma sequência dos principais passos que devem ser realizados para um processo de aprendizagem de máquina utilizando regressão linear.

Algoritmo 2.1 Regressão Linear

Importar base de dados D
Decompor D entre X e $\mathbf{y}$
Codificar colunas em X que possuam dados categóricos
Remover uma coluna de cada variável categórica codificada em X
Normalizar dados em X
Dividir X e $\mathbf{y}$ entre dados de treinamento, dados de validação e dados de teste
Utilizando os dados de treinamento, encontrar o vetor $\mathbf{p}$ através da equação: $X^T\mathbf{y} - X^T X\mathbf{p} = \mathbf{0}$
Para uma nova observação cujo índice é $j + 1$, encontrar $\hat{y}_{j+1}$ através da equação: $\mathbf{x}_{j+1}\ \mathbf{p} = \hat{y}_{j+1}$

2.4 Redução de dimensionalidade de matrizes

Diferentemente de algoritmos baseados em conteúdo, onde o conjunto de dados é organizado de forma que as variáveis preditivas e o atributo alvo são claramente demarcados (vide Figura 4), a construção de algoritmos que lidam com filtragem colaborativa requer

que o conjunto de dados assuma a forma de uma matriz $A_{m \times n}$ relativa a m usuários e n itens, onde cada elemento a_{ij} é a avaliação dada pelo usuário i ao item j.

Nas aplicações de algoritmos de aprendizado de máquina em situações reais, a matriz de avaliações $A_{m \times n}$ frequentemente assume grandes dimensões (vide caso da *Amazon*, com milhões de usuários e milhões de itens disponíveis na plataforma). Nesses casos, é comum que uma boa parte das linhas e colunas dessa matriz seja altamente correlacionada, uma vez que o padrão de consumo de muitos usuários tende a ser parecido. Dessa forma, a matriz original possui um alto grau de informações redundantes, e grande parte da informação presente na matriz A pode ser aproximada através de uma decomposição de matrizes de posto reduzido $p < min(m, n)$, ou seja, podemos reduzir o número de dimensões analisadas e manter as informações mais importantes da matriz A utilizando um conjunto menor de dados.

Essa redução de dimensionalidade só é possível uma vez que as correlações entre os itens e/ou usuários na matriz A podem ser explicadas através de *fatores latentes*, ou seja, características implícitas ao conjunto de dados e que explicam a maior parte de sua variância.

A redução de dimensionalidade pode ser aplicada no contexto de sistemas de recomendação de duas formas principais, detalhadas abaixo (AGGARWAL, 2016):

1. Na criação de uma representação da matriz de avaliações, reduzindo a dimensionalidade de itens ou de usuários. Essa técnica é utilizada para mitigar o problema de matrizes de avaliações esparsas, comumente encontradas em casos reais quando nem todos os usuários avaliam todos os itens;

2. Na criação de uma representação da matriz de avaliações, porém determinando de maneira simultânea as representações latentes tanto de itens como de usuários. Nesse caso, a matriz de avaliações é completada de uma vez só, sem a necessidade de implementar nenhum algoritmo adicional.

Iremos iniciar esse capítulo com uma seção sobre a intuição geométrica sobre os fatores latentes, seguida de uma discussão sobre a conexão entre fatoração de matrizes, fatores latentes, e redução de dimensionalidade. Posteriormente abordaremos o uso da redução de dimensionalidade no contexto de sistemas de recomendação, tanto como ferramenta de mitigação do problema de matrizes esparsas como o seu uso exclusivo para completar uma matriz de avaliação incompleta.

2.4.1 Embasamento matemático

2.4.1.1 Fatores latentes: Intuição geométrica

Considere uma matriz $A_{m\times n}$. A fim de exemplificar o conceito, trabalharemos momentâneamente com $m = 20$ e $n = 2$, ou seja, uma matriz com avaliações de vinte usuários a dois itens específicos, A e B. Suponha também que as avaliações dos dois itens sejam correlacionadas positivamente. Abaixo apresentamos o gráfico de dispersão de duas dimensões, onde o eixo das abcissas representa as avaliações dadas ao item A, e o eixo das ordenadas as do item B.

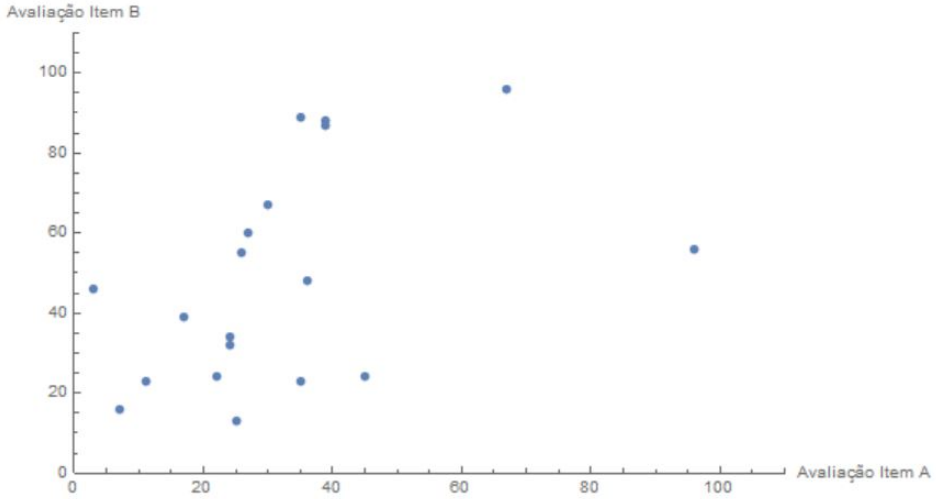

Figura 7 – Gráfico de dispersão para vinte avaliações de dois produtos. Fonte: autora.

Graças a disposição dos dados, o gráfico de dispersão 7 pode ser aproximado por uma reta no sentido no qual os dados estão mais "alongados", conforme abaixo:

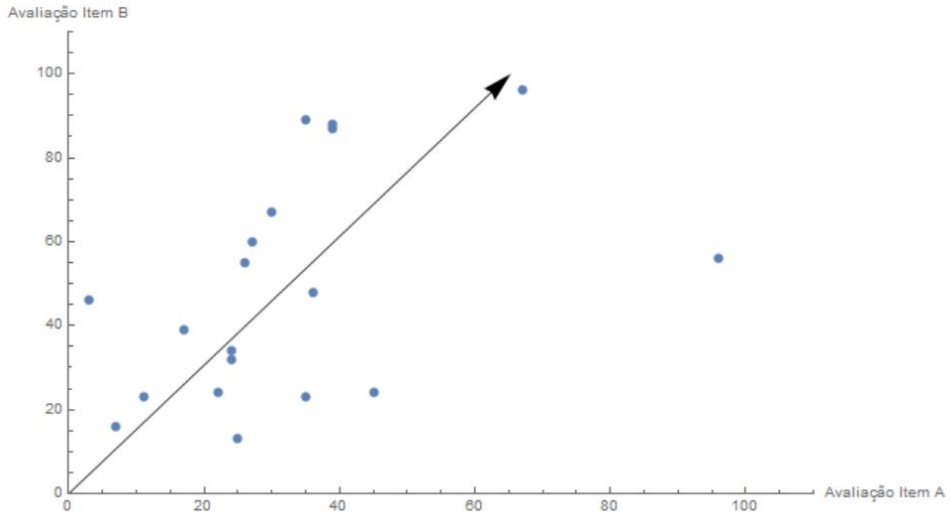

Figura 8 – Gráfico de dispersão para vinte avaliações de dois produtos, aproximado por uma reta. Fonte: autora.

A reta representada na Figura 8 é um vetor latente que explica o sentido de maior variância nas avaliações dos itens A e B. Através da aproximação dada por esse vetor latente, é possível representar os pontos $(Avaliação_A, Avaliação_B)$ como uma só dimensão, indicando apenas a localização de cada ponto na reta. Através desse sistema, podemos representar dados bidimensionais de maneira unidimensional, reduzindo a dimensionalidade do conjunto de dados inicial. Nesse exemplo, a aproximação dos dados por uma reta significa que a matriz de dados original possui posto $p = 1$ (AGGARWAL, 2016).

Além disso, suponha que tenhamos as coordenadas da reta que aproxima o conjunto de dados e a avaliação para apenas um dos itens. Podemos aproximar a avaliação para o segundo item ao calcular a interseção dessa reta com um hiperplano de dimensão p (que no caso desse exemplo é uma reta unidimensional). De fato, as avaliações desconhecidas em um conjunto de dados podem ser aproximadas se tivermos pelo menos p avaliações conhecidas para o usuário em questão (AGGARWAL, 2016). Veja que, nesse exemplo, se a avaliação do item B é 60 para um usuário específico, a avaliação do item A pode ser aproximada como 39, conforme exemplificado pela figura abaixo.

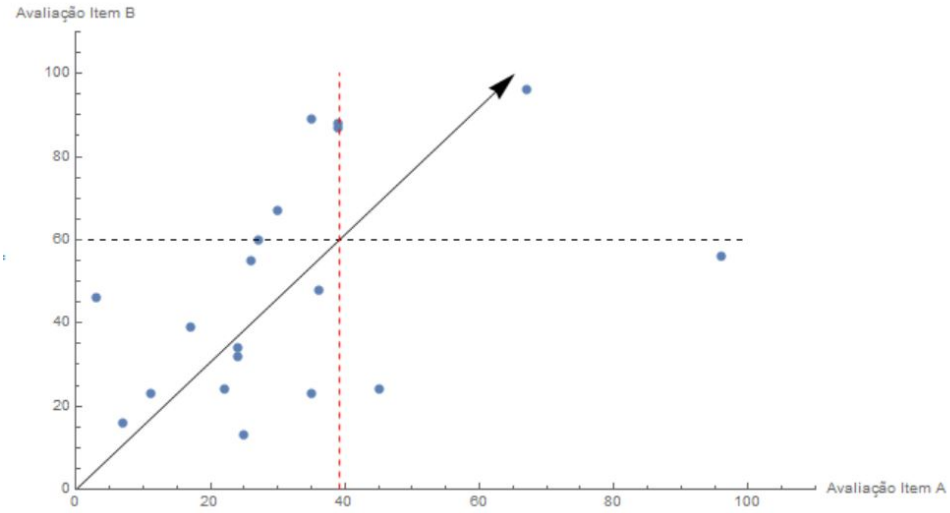

Figura 9 – Estimação da avaliação de A através de uma aproximação unidimensional do conjunto de dados. Fonte: autora.

É importante notar que métodos de redução de dimensionalidade baseados em fatores latentes são construídos a partir de correlações e redundâncias nos dados. Se o conjunto de dados não possui nenhum tipo de redundância não é possível encontrar fatores latentes e portanto esses métodos não podem ser utilizados.

2.4.1.2 Fatoração de matrizes e fatores latentes

Uma das formas de entender e construir a redução de dimensionalidade de matrizes é através de sua fatoração. Considere uma matriz $A_{m\times n}$ cujos elementos são todos

conhecidos. Seja $p < min(m, n)$ o posto dessa matriz. A matriz A pode ser aproximada através da decomposição:

$$A \approx U_p \Sigma_p V_p^T \tag{2.10}$$

Onde U_p é uma matriz $m \times p$, Σ_p é uma matriz $p \times p$ e V_p é uma matriz $n \times p$. Realizando a multiplicação entre U_p e Σ_p simplificamos a equação 2.10, conforme abaixo:

$$A \approx U_p' V_p^T \tag{2.11}$$

Onde U_p' continua sendo uma matriz $m \times p$ (apesar de possuir elementos com valores diferentes de U_p) e V_p não é modificada. Cada coluna de U_p' representa um dos p vetores que formam uma base para o espaço coluna de A, e cada linha j de V_p possui os coeficientes que, quando combinados aos vetores coluna de U_p', se transformam na coluna j de A. Podemos também considerar as colunas de V_p como os vetores que formam a base para o espaço linha de A e as linhas de U_p' como os coeficientes correspondentes (AGGARWAL, 2016).

Trazendo esse conceito ao universo de sistemas de recomendação, cada linha i da matriz U_p' consiste em um vetor $\mathbf{u_i} \in R^p$ que nos diz a tendência do usuário i em A de se identificar com cada um dos p fatores latentes. Da mesma forma, cada coluna j em V_p^T consiste em um vetor $\mathbf{v_j} \in R^p$ que explica a relação entre um item j e cada um dos fatores latentes. Com isso em mente, a estimativa da nota a_{ij} da avaliação de um usuário i ao item j pode ser facilmente encontrada através do produto interno entre a linha de U_p' referente ao usuário i e a coluna de V_p^T referente ao item j, conforme equação 2.12:

$$a_{ij} = \mathbf{v_j}^T \ \mathbf{u_i} \tag{2.12}$$

Existem vários métodos de fatoração de matrizes que permitem a construção da fatoração da matriz A como detalhado anteriormente, e entre eles está a decomposição por valores singulares, que será discutida abaixo. Uma vez com a base teórica da fatoração esclarecida, vamos seguir para definir a relação entre fatoração de matrizes e redução de dimensionalidade através de fatores latentes.

2.4.1.3 *Decomposição em valores singulares*

Seja A uma matriz $m \times n$. Sabemos que, se $m = n$ e A possuir uma base de autovetores, A pode ser decomposta em três matrizes M, Λ e M^{-1}, onde M é a matriz cujas

colunas são os autovetores de A e Λ é uma matriz diagonal cuja diagonal é composta pelos autovalores de A (BOLDRINI et al., 1986). Essa decomposição é chamada de decomposição espectral.

Agora, no caso em que $m \neq n$ ou que não seja possível encontrar uma base de autovetores de A, ainda assim é possível decompor a matriz A através da decomposição em valores singulares. Nesse caso, A será decomposta nas três matrizes U, Σ e V^T, onde U é a matriz cujas colunas são os autovetores de AA^T, Σ é a matriz diagonal cuja diagonal é composta pela raiz quadrada dos autovalores de AA^T (que também são autovalores de A^TA), e V é a matriz cujas colunas são os autovetores de A^TA (vide equação 2.13 abaixo).

$$A = U\Sigma V^T \tag{2.13}$$

Para fins de demonstração, seja A uma matriz $m \times n$ com posto p. É evidente que A^TA é uma matriz $n \times n$ e simétrica. Além disso, A^TA é semi-definida positiva (GUNDERSEN, 2018), como demonstrado abaixo:

$$\begin{aligned} G &= A^TA \\ \mathbf{x}^TG\mathbf{x} &= \mathbf{x}^TA^TA\mathbf{x} \\ \mathbf{x}^TG\mathbf{x} &= (A\mathbf{x})^TA\mathbf{x} \\ \mathbf{x}^TG\mathbf{x} &= ||A\mathbf{x}||^2 \\ \mathbf{x}^TG\mathbf{x} &\geqslant 0 \end{aligned} \tag{2.14}$$

Como uma matriz simétrica, podemos afirmar que A^TA admite uma base ortonormal de autovetores, e pode então ser decomposta conforme abaixo:

$$\begin{aligned} A^TA &= V\Lambda V^{-1} \\ A^TA &= V\Lambda V^T \end{aligned} \tag{2.15}$$

Onde a segunda igualdade acima se deve ao fato de que V é a matriz $n \times n$ cujas colunas são os autovetores ortonormais de A^TA, e como uma matriz ortormal a sua inversa é igual à transposta. Como mencionado anteriormente, Λ é a matriz diagonal $n \times n$ cujos elementos não nulos são os autovalores de A^TA.

Como A^TA é semi-definida positiva, todos os seus autovalores são maiores ou iguais a zero. Ao considerarmos a raiz quadrada desses autovalores, encontramos o que chamamos de **valores singulares**, costumariamente denotados pela letra σ (GUNDERSEN, 2018).

Sabemos então que, para um autovetor $\mathbf{v_i} \in R^n$ qualquer de A^TA:

$$A^TA\mathbf{v_i} = \lambda_i\mathbf{v_i} = \sigma_i^2\mathbf{v_i} \tag{2.16}$$

Vamos definir um vetor $\mathbf{u_i} \in R^m$ de forma que $\mathbf{u_i} = \dfrac{A\mathbf{v_i}}{\sigma_i}$. Por construção, $\mathbf{u_i}$ será um autovetor de AA^T, conforme demonstrado abaixo:

$$\begin{aligned}
AA^T\mathbf{u_i} &= AA^T\frac{A\mathbf{v_i}}{\sigma_i} \\
AA^T\mathbf{u_i} &= AA^TA\mathbf{v_i}\frac{1}{\sigma_i} \\
AA^T\mathbf{u_i} &= A\sigma_i^2\mathbf{v_i}\frac{1}{\sigma_i} \\
AA^T\mathbf{u_i} &= \sigma_i^2\frac{A\mathbf{v_i}}{\sigma_i} \\
AA^T\mathbf{u_i} &= \lambda_i\mathbf{u_i}
\end{aligned} \tag{2.17}$$

Onde a terceira igualdade acima decorre de que $\mathbf{v_i}$ é autovetor de A^TA, então $A^TA\mathbf{v_i} = \lambda_i\mathbf{v_i} = \sigma_i^2\mathbf{v_i}$.

Além disso, $\mathbf{u_i}$ é um vetor unitário, como provamos abaixo.

$$\begin{aligned}
\|\mathbf{u_i}\| &= 1 \\
\sqrt{u_{1i}^2 + u_{2i}^2 + ... + u_{mi}^2} &= 1 \\
u_{1i}^2 + u_{2i}^2 + ... + u_{mi}^2 &= 1 \\
\mathbf{u_i^T u_i} &= 1 \\
\frac{A\mathbf{v_i}}{\sigma_i}^T\frac{A\mathbf{v_i}}{\sigma_i} &= 1 \\
\frac{\mathbf{v_i}^TA^T}{\sigma_i}\frac{A\mathbf{v_i}}{\sigma_i} &= 1 \\
\frac{\mathbf{v_i}^TA^TA\mathbf{v_i}}{\sigma_i^2} &= 1 \\
\frac{\mathbf{v_i}^T\sigma_i^2\mathbf{v_i}}{\sigma_i^2} &= 1 \\
\mathbf{v_i}^T\mathbf{v_i} &= 1 \\
1 &= 1
\end{aligned} \tag{2.18}$$

Uma vez que $\mathbf{v_i}$ é um vetor ortonormal.

Concatenando os vetores $\mathbf{u_i}$ para todo $i = 1, ..., m$, formamos a matriz $U_{m\times m}$, cujas colunas são formadas pelos vetores $\mathbf{u_i}$. Temos que U será igual ao produto das matrizes $A_{m\times n}$, $V_{n\times n}$ e $\Sigma^{-1}_{n\times m}$, onde $\Sigma_{m\times n}$ é a matriz diagonal cujos elementos não nulos são as raizes quadradas dos autovalores de A^TA, que também são autovalores de AA^T, uma vez que ambas as matrizes possuem os mesmos autovalores diferentes de zero. Com isso temos que:

$$\begin{aligned} U &= AV\Sigma^{-1} \\ U\Sigma &= AV\Sigma^{-1}\Sigma \\ U\Sigma &= AV \\ U\Sigma V^{-1} &= AVV^{-1} \\ U\Sigma V^T &= A \end{aligned} \tag{2.19}$$

Uma vez que V é ortonormal e sua inversa é igual a transposta (GUNDERSEN, 2018).

Provamos então que qualquer matriz $A_{m\times n}$ pode ser decomposta nas matrizes $U_{m\times m}$, $\Sigma_{m\times n}$ e $V^T_{n\times n}$.

Convencionalmente, os autovetores em U e V são ordenados conforme seus valores singulares correspondentes em Σ, de maneira decrescente.

2.4.1.4 *Redução de dimensionalidade utilizando decomposição em valores singulares*

Como discutido anteriormente, graças ao alto grau de informações redundantes na matriz de avaliações $A_{m\times n}$, grande parte da informação presente na matriz pode ser aproximada através de uma decomposição de matrizes de posto $p < min(m, n)$. O objetivo é reduzir o número de dimensões analisadas sem perder as informações mais importantes da matriz A.

A fim de entender qual a perda de informação relativa à redução de uma ou mais dimensões, calculamos a matriz de covariância de A. A covariância é a medida do grau de interdependência entre duas variáveis, ou seja, caso duas variáveis sejam altamente correlacionadas, a covariância será alta. A covariância será nula caso as duas variáveis sejam independentes entre si (HUI, 2019).

A matriz de covariância será uma matriz simétrica, cuja diagonal é a variância de uma variável com relação à ela mesma.

A fim de ilustrar a aplicação dessa decomposição no contexto de sistemas de recomendação, tomemos como exemplo a matriz $A_{m\times n}$ de m usuários e n itens, tal que o elemento a_{ij} é a nota dada pelo usuário i ao item j. Vamos centralizar a matriz A

subtraindo de cada elemento a média de todas as notas dadas àquele item (média dos elementos na coluna j), de forma que a média dos valores em cada coluna é zero. Para variáveis centralizadas, a covariância entre duas variáveis **a** e **b** pode ser calculada através do valor esperado da multiplicação entre elementos das duas variáveis, conforme equação 2.20:

$$cov(\mathbf{a}, \mathbf{b}) = E[\mathbf{ab}] \tag{2.20}$$

E a variância de uma variável com relação a ela mesma é calculada da mesma forma:

$$var(\mathbf{a}) = cov(\mathbf{a}, \mathbf{a}) = E[\mathbf{aa}] \tag{2.21}$$

Reescrevendo as equações 2.20 e 2.21 na forma matricial para n variáveis com m elementos cada, é possível encontrar a matriz de covariância de A, denotada abaixo por C_A (HUI, 2019):

$$C_A = \frac{1}{n} A^T A \tag{2.22}$$

Sabemos que qualquer matriz A pode ser decomposta utilizando a decomposição por valores singulares. Seja então $A = U\Sigma V^T$:

$$\begin{aligned}
C_A &= \frac{1}{n}(U\Sigma V^T)^T(U\Sigma V^T) \\
C_A &= \frac{1}{n}(V\Sigma^T U^T)(U\Sigma V^T) \\
C_A &= \frac{1}{n} V\Sigma^T (U^T U)\Sigma V^T \\
C_A &= \frac{1}{n} V\Sigma^T \Sigma V^T \\
C_A &= \frac{1}{n} V\Sigma^2 V^T \\
C_A &= V\frac{\Sigma^2}{n} V^T \\
C_A &= VDV^T
\end{aligned} \tag{2.23}$$

Onde D acima é a matriz $\frac{\Sigma^2}{n}$. Como os elementos da diagonal da matriz Σ são os valores singulares da matriz A e que por sua vez são convencionalmente ordenados de forma decrescente, podemos afirmar que os valores singulares de maior valor estão

diretamente relacionados às variáveis de A que possuem a maior variância, ou seja, que detém o maior nível de informação.

De fato, uma das propriedades da matriz de covariância é que a variância de A é o traço de C_A, que consequentemente é o traço da matriz diagonal D, como demonstrado abaixo (OLIVEIRA, 2016):

$$\begin{aligned} traço(C_A) &= traço(VDV^T) \\ traço(C_A) &= traço(DVV^T) \\ traço(C_A) &= traço(DI) \\ traço(C_A) &= traço(D) \end{aligned} \tag{2.24}$$

Como $D = \frac{\Sigma^2}{n}$, o que decorre da afirmação acima é que podemos calcular a proporção da variância de A relativa a cada valor singular σ_i através da razão entre σ_i^2 e a soma dos quadrados dos valores singulares em Σ. Dessa forma, sabendo os valores singulares da matriz A, é possível eliminar as variáveis redundantes - ou seja, aquelas que possuem os menores valores singulares e consequentemente representam pouca variância dos dados - e reconstruir a matriz A^*, que é a aproximação da matriz A, através da multiplicação das matrizes U_p, Σ_p e V_p truncadas após a remoção de variáveis redundantes. A idéia é manter apenas os vetores latentes do conjunto de dados. Além disso, é possível utilizar um processo iterativo para estimar os elementos faltantes na matriz A em situações onde A não é completamente especificada (AGGARWAL, 2016).

Voltemos ao exemplo inicial da matriz $A_{20\times2}$ com as avaliações de vinte usuários a dois itens A e B. Encontramos as duas matrizes quadradas AA^T e A^TA e seus respectivos autovalores não nulos, que são $79.668,4$ e $5.339,6$. Com isso, podemos calcular a proporção p_{σ_i} da variância dos dados relativa a cada valor singular σ_i de A, conforme abaixo:

$$\begin{aligned} p_{\sigma_1} &= \frac{79.668,4^2}{(79.668,4^2 + 5.339,6^2)} \\ p_{\sigma_1} &= 0,996 \end{aligned}$$

$$\begin{aligned} p_{\sigma_2} &= \frac{5.339,6^2}{(79.668,4^2 + 5.339,6^2)} \\ p_{\sigma_2} &= 0,005 \end{aligned}$$

Veja que o vetor latente relativo ao primeiro valor singular representa 99,6% da variância nos dados de A. Com isso, podemos representar a matriz A como uma matriz

de posto $p = 1$, removendo as colunas e linhas de U, Σ e V relativas ao segundo valor singular em Σ. A nova matriz A^* será representada a partir da multiplicação entre as matrizes $U_{20\times1}$, $\Sigma_{1\times1}$ e $V_{2\times1}$.

Agora veja que, realizando os cálculos, o vetor linha em ${V_{2\times1}}^T$ é o vetor $(0, 653, 1)$. Esse é o vetor latente representado nas Figuras 8 e 9 como a reta que melhor aproxima o conjunto de dados em A.

2.4.2 Aplicação em sistemas de recomendação

Como comentado anteriormente, a redução de dimensionalidade em sistemas de recomendação pode ser aplicada de duas formas principais: como ferramenta para mitigar o problema de matrizes esparsas ou como forma de prever avaliações desconhecidas de usuários a itens. Abaixo abordaremos ambas as aplicações em um contexto de sistemas de recomendação.

2.4.2.1 Mitigação do problema de matrizes esparsas

O objetivo do uso de redução de dimensionalidade para mitigar o problema de matrizes esparsas é o de redimensionar uma matriz esparsa $A_{m\times n}$ em uma matriz $R_{m\times p}$, com $p < n$, de forma que $R_{m\times p}$ seja completamente especificada. A partir dessa nova matriz é possível por exemplo estabelecer vizinhanças entre usuários (grupos de usuários similares entre si) ou criar árvores de decisão, sem o problema de dados desconhecidos.

O primeiro passo a fim de construir uma representação de menor dimensão da matriz A é calcular a matriz de covariância $n \times n$ entre as colunas de A, a qual denominaremos C_A. Como A é uma matriz esparsa, o cálculo de C_A é feito utilizando apenas as linhas referentes aos usuários que possuem avaliações para os dois itens em questão. No caso em que não existem usuários em comum entre dois itens, a covariância entre esses itens é considerada zero (AGGARWAL, 2016).

Uma vez com a matriz de covariância construída, aplicamos a decomposição espectral em C_A conforme abaixo:

$$C_A = VDV^T \tag{2.25}$$

Encontramos os p autovalores responsáveis pela maior parte da variância no conjunto de dados, conforme discussão na seção anterior. Seja V_p a matriz $n \times p$ resultante da eliminação das $n - p$ colunas de V que menos contribuem com a variância nos dados.

Iremos calcular a contribuição de cada avaliação observada em A na projeção em cada vetor latente de V_p, e então encontrar a média dessas contribuições para cada

item. Seja g_i a i-ésima coluna da matriz V_p. Ao j-ésimo elemento de g_i chamaremos g_{ji}. Seja a_{uj} um elemento conhecido da matriz esparsa de avaliações A, sendo a_{uj} a avaliação relativa ao usuário u e o item j. Temos então que a contribuição de a_{uj} no vetor latente g_i é a multiplicação $a_{uj}g_i$. Se definirmos o conjunto I_u como o conjunto de todas as avaliações conhecidas para o usuário u, a contribuição média r_{ui} do usuário u no vetor latente g_i é calculada através da equação 2.26 abaixo (AGGARWAL, 2016):

$$r_{ui} = \frac{\sum_{j \in I_u} r_{uj} g_{ji}}{|I_u|} \tag{2.26}$$

Uma vez com r_{ui} calculado para todos os m usuários e todos os p vetores latentes, construimos a matriz $R_{m \times p}$, tal que $R = [r_{ui}]_{m \times p}$. A partir dessa nova matriz R completamente especificada podem ser aplicados algoritmos como k-vizinhos mais próximos ou árvores de decisão, métodos esses que serão abordados nas próximas sessões.

2.4.2.2 Previsão de avaliações

Pode-se também utilizar a redução de dimensionalidade em sistemas de recomendação de maneira exclusiva, sem aplicar o uso de qualquer outro algoritmo. O objetivo aqui é utilizar os fatores latentes em um conjunto de dados de forma a calcular a previsão de uma avaliação a_{ij} inicialmente desconhecida.

Exemplo 2. ***Redução de dimensionalidade para previsão de avaliações***

Seja A uma matriz $m \times n$ gerada a partir das notas dadas por m usuários a n produtos diferentes. Suponha que as notas variem de 1 a 5, de forma discreta. Como nem todos os usuários deram notas a todos os produtos, alguns elementos da matriz A estão faltando. Abaixo é exemplificada a matriz A para uma amostra de 5 usuários e 3 produtos.

$$A = \begin{bmatrix} 2 & 2 & ? \\ 4 & 3 & 1 \\ ? & 4 & 5 \\ 2 & 1 & 4 \\ 4 & 5 & 2 \end{bmatrix}$$

Queremos encontrar valores para os dois elementos faltantes, mais especificamente os elementos a_{31} e a_{13}, onde a_{ij} representa a nota dada pelo usuário i ao item j. Uma vez com a matriz completa, podemos usar os valores de a_{31} e a_{13} para decidir se é interessante recomendar o item 1 ao terceiro usuário e o item 3 ao primeiro usuário.

Vamos inicialmente preencher os valores desconhecidos em A com a média de cada coluna, gerando a matriz A_1.

$$A_1 = \begin{bmatrix} 2 & 2 & 3 \\ 4 & 3 & 1 \\ 3 & 4 & 5 \\ 2 & 1 & 4 \\ 4 & 5 & 2 \end{bmatrix}$$

Aplicando a decomposição em valores singulares em A_1 *obtemos as matrizes* U, Σ *e* V, *como demonstrado abaixo. Temos então que* $A_1 = U \times \Sigma \times V^T$, *onde:*

$$U = \begin{bmatrix} 0,337 & 0,234 & -0,042 & 0,775 & -0,479 \\ 0,389 & -0,502 & -0,648 & -0,258 & -0,332 \\ 0,579 & 0,347 & 0,435 & -0,516 & -0,295 \\ 0,334 & 0,561 & -0,517 & 0 & 0,553 \\ 0,536 & -0,508 & 0,348 & 0,258 & 0,516 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 11,90 & 0 & 0 \\ 0 & 3,82 & 0 \\ 0 & 0 & 1,37 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0,568 & 0,601 & 0,562 \\ -0,368 & -0,425 & 0,827 \\ -0,736 & 0,676 & 0,020 \end{bmatrix}$$

Percebe-se que o primeiro valor singular de 11,90 representa $89,6\%$ *da variância em* A_1, *conforme demonstrado abaixo:*

$$\frac{11,90^2}{11,90^2 + 3,82^2 + 1,37^2} = 89,6\%$$

Se tomarmos o primeiro e o segundo valor singular, eliminando o terceiro, contamos com $98,8\%$ *da variância. Vamos eliminar o terceiro valor singular de 1,37, e reconstruir a matriz* A_1 *utilizando* $U_{5\times2}$, $\Sigma_{2\times2}$ *e* $V_{3\times2}$, *de forma a encontrar* A_2*:*

$$U_{5\times2} = \begin{bmatrix} 0,337 & 0,234 \\ 0,389 & -0,502 \\ 0,579 & 0,347 \\ 0,334 & 0,561 \\ 0,536 & -0,508 \end{bmatrix}$$

$$\Sigma_{2\times 2} = \begin{bmatrix} 11,90 & 0 \\ 0 & 3,82 \end{bmatrix}$$

$$V^T_{2\times 3} = \begin{bmatrix} 0,568 & 0,601 & 0,562 \\ -0,368 & -0,425 & 0,827 \end{bmatrix}$$

Como $A_2 = U_{5\times 2} \times \Sigma_{2\times 2} \times V^T_{2\times 3}$, *temos que* A_2 *é:*

$$A_2 = \begin{bmatrix} 1,95 & 2,03 & 2,99 \\ 3,33 & 3,59 & 1,01 \\ 3,43 & 3,60 & 4,96 \\ 1,47 & 1,48 & 4,00 \\ 4,34 & 4,66 & 1,98 \end{bmatrix}$$

Veja que apesar da redução de dimensionalidade a variação entre os valores de A_1 *e* A_2 *é pequena. Embora a decomposição original tenha dimensionalidade 3 (três valores singulares), podemos realizar a decomposição truncada com dimensão 2 (sem o menor valor singular) iterativamente de forma a completar a matriz A original e encontrar os valores de* a_{13} *e* a_{31}*. O benefício de realizar a decomposição truncada é a de que dessa forma o sistema requer menos esforço computacional, o que é muito importante em casos onde existem milhões de usuários e milhões de itens.*

Vamos substituir os elementos a_{13} *e* a_{31} *que eram desconhecidos na matriz A original pelos valores em* A_2*, ou seja, os valores encontrados após a decomposição truncada em duas dimensões:*

$$A_2 = \begin{bmatrix} 2 & 2 & 2,99 \\ 4 & 3 & 1 \\ 3,43 & 4 & 5 \\ 2 & 1 & 4 \\ 4 & 5 & 2 \end{bmatrix}$$

Aplicando a decomposição truncada com posto $p = 2$ *novamente, chegamos na seguinte matriz* A_3*:*

$$A_3 = \begin{bmatrix} 2,03 & 1,98 & 2,99 \\ 3,33 & 3,59 & 1,05 \\ 3,75 & 3,71 & 4,97 \\ 1,57 & 1,38 & 4,03 \\ 4,37 & 4,66 & 1,96 \end{bmatrix}$$

Veja que os valores de a_{13} e a_{31} pouco mudaram de A_2 para A_3. A idéia é realizar esse processo iterativamente para A_i, até que a diferença entre os valores desconhecidos entre iterações subsequentes seja muito pequena, de forma que esses valores se estabilizem. Baseado nisso, podemos decidir ou não por sugerir esses itens aos usuários em questão.

O Algoritmo 2.2 resume os passos para utilizar a decomposição em valores singulares para a mitigação do problema de matrizes esparsas. De maneira similar, o Algoritmo 2.3 define as etapas necessárias para utilização da decomposição em valores singulares a fim de prever avaliações.

Algoritmo 2.2 Decomposição em valores singulares para mitigação do problema de matrizes esparsas

Importar matriz de avaliações A
Encontrar matriz de covariância C_A
Aplicar decomposição espectral em C_A e definir os p autovalores responsáveis pela maior parte da variância
Definir V_p como a matriz com apenas os p maiores vetores latentes
Calcular a contribuição média do usuário u no vetor latente g_i através da equação:

$$r_{ui} = \frac{\sum_{j \in I_u} r_{uj}\ g_{ji}}{|I_u|} \tag{2.27}$$

Construir a matriz $R = [r_{ui}]_{m \times p}$ completamente especificada

Algoritmo 2.3 Decomposição em valores singulares para previsão de avaliações

Importar matriz de avaliações A
Preencher valores não especificados com a média de cada coluna em A
Encontrar a matriz de valores singulares Σ
Definir o número k de dimensões a serem reduzidas em A de acordo com a variância dos valores singulares em Σ
Remover da matriz Σ as linhas e colunas referentes aos k valores singulares definidos no passo anterior, assim como suas respectivas colunas em U e linhas em V^T
Encontrar a matriz A^* através da multiplicação de U, Σ e V^T truncados após redução de k dimensões
Substituir os valores não especificados de A pelos seus respectivos valores em A^*, e aplicar a decomposição truncada novamente. Seguir esse processo iterativamente até a convergência dos valores inicialmente não especificados

2.5 K-vizinhos mais próximos

O método dos k-vizinhos mais próximos, também conhecido como KNN do inglês *k-nearest neighbors*, se utiliza da distância entre instâncias para fazer as previsões. Nesse método, são criadas vizinhanças para cada instância, e a estimativa do resultado de uma instância específica é calculada com base nos resultados de sua vizinhança.

Observe, por exemplo, a Figura 10. Seja o usuário em vermelho o usuário para o qual gostaríamos de predizer uma certa avaliação. Definimos então uma vizinhança para esse usuário como sendo o grupo de k usuários mais próximos a ele (grupo de usuários em amarelo na figura). As predições do algoritmo serão baseadas no comportamento dos usuários que fazem parte dessa vizinhança.

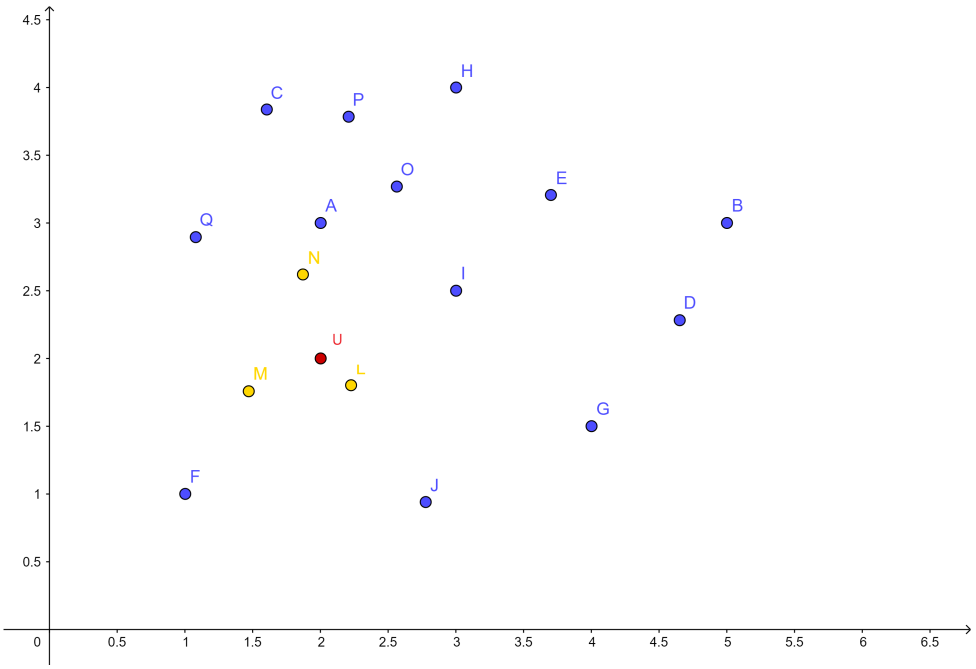

Figura 10 – Exemplo de vizinhança para um usuário. Fonte: autora

2.5.1 Embasamento matemático

O método dos k-vizinhos mais próximos pode ser utilizado tanto em análises de regressão como de classificação, sendo que essas duas aplicações seguem costumariamente uma abordagem de sistemas baseados em conteúdo. Além disso, o método pode ser também aplicado em sistemas de filtragem colaborativa baseados em vizinhança, onde é possível calcular a similaridade entre usuários e/ou itens a fim de prever uma avaliação desconhecida.

2.5.1.1 Regressão

Seja a base de dados D um conjunto de m tuplas $\{\mathbf{x_i}\ ,\ y_i\}$, i = 1, ..., m, onde m é o número total de observações, $\mathbf{x_i} \in R^n$, sendo n o número de variáveis de previsão, e $y_i \in R$. Seja $\mathbf{x_t}$ a nova observação para a qual queremos encontrar uma previsão $\hat{y}_t$.

Em um primeiro momento, calcula-se a distância d_i entre o vetor $\mathbf{x_t}$ e cada vetor $\mathbf{x_i}$, $i = 1, 2, ..., m$. O cálculo da distância pode ser feito de várias formas, sendo as mais comuns a distância euclidiana e o ângulo entre vetores.

Dado um certo $k \in Z$, com $k > 0$, são selecionados os k vetores $\mathbf{x_i}$ mais próximos a $\mathbf{x_t}$ (daí o nome do método, k-vizinhos mais próximos), ou seja, os k vetores $\mathbf{x_i}$ com a menor distância d_i. Para encontrar a previsão $\hat{y}_t$, usa-se a média ponderada pela distância dos valores de y_i para todo $\mathbf{x_i} \in N(\mathbf{x_t}, k)$, sendo $N(\mathbf{x_t}, k)$ o conjunto gerado pelos k vetores mais próximos a $\mathbf{x_t}$.

$$\hat{y}_t = \frac{\sum\limits_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} w_i y_i}{\sum\limits_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} w_i} \tag{2.28}$$

Onde w_i é a função peso. Como queremos dar um peso menor a vetores mais distantes de $\mathbf{x_t}$, a função peso deve ser inversamente proporcional à distância d_i. Caso d_i seja o ângulo entre $\mathbf{x_i}$ e $\mathbf{x_t}$, podemos assumir $w_i = \cos d_i$, uma vez que quanto maior o ângulo, menor o cosseno do ângulo entre os vetores. Caso d_i seja a distância euclidiana entre $\mathbf{x_i}$ e $\mathbf{x_t}$, é possível determinar $w_i = \frac{1}{d_i^2}$, por exemplo (KANE, 2018).

2.5.1.2 Classificação

Seja a base de dados D um conjunto de m tuplas $\{\mathbf{x_i}\ , y_i\}$, i = 1, ..., m, onde m é o número total de observações, $\mathbf{x_i} \in R^n$, sendo n é o número de variáveis de previsão, e y_i é uma variável categórica pertencente a um conjunto de classes $C = \{c_1, c_2, ...c_h\}$, onde h o número total de classes. Seja $\mathbf{x_t}$ a nova observação para a qual queremos encontrar uma previsão $\hat{y}_t$, ou seja, queremos prever qual será a classe de $\mathbf{x_t}$.

Da mesma maneira que na análise de regressão, calcula-se primeiro a distância d_i entre o vetor $\mathbf{x_t}$ e cada vetor $\mathbf{x_i}$, $i = 1, 2, ..., m$.

Dado um certo $k \in Z$, com $k > 0$, são selecionados os k vetores $\mathbf{x_i}$ com a menor distância d_i. Para encontrar a previsão $\hat{y}_t$, verificamos a qual classe pertence cada $\mathbf{x_i} \in N(\mathbf{x_t}, k)$, sendo $N(\mathbf{x_t}, k)$ o conjunto gerado pelos k vetores mais próximos a $\mathbf{x_t}$. A previsão $\hat{y}_t$ será a classe c_j que irá maximizar a equação 2.29 abaixo:

$$\hat{y}_t = \underset{c_j \in \{c_1, c_2, ..., c_h\}}{\operatorname{argmax}} \sum_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} F(y_i, c_j) \tag{2.29}$$

Onde $F(y_i, c_j) = 1$ se $y_i = c_j$ e $F(y_i, c_j) = 0$ caso contrário.

O operador *argmax* faz referência aos argumentos que maximizam a função $\sum\limits_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} f(y_i, c)$. Em outras palavras, a função 2.29 procura retornar a classe c_j que maximiza o somatório, que nada mais é do que a classe mais frequente no conjunto $N(\mathbf{x_t}, k)$ (BATISTA; SILVA, 2009).

2.5.1.3 Modelos de vizinhança

Em sistemas de filtragem colaborativa baseados em vizinhança, onde possuímos uma matriz $A_{m\times n}$ com as avaliações de m usuários a n itens, cada linha relativa a um usuário (ou coluna relativa a um item) é vista como um vetor $\mathbf{x}$. Calcula-se a distância do vetor relativo ao usuário (ou item) para o qual queremos prever a avaliação (vetor $\mathbf{x_i}$) e todos os outros vetores $\mathbf{x_u}$.

Trabalhemos a seguir com o cenário de similaridade entre usuários. Seja $P_i(j)$ o conjunto dos k usuários mais próximos ao usuário i cujas avaliações para o item j são conhecidas. Note que w_{ui} é a função similaridade entre os vetores correspondentes ao usuário u e o usuário i. Podemos encontrar a previsão da avaliação a_{ij} do usuário i ao item j conforme abaixo:

$$a_{ij} = \frac{\sum\limits_{u\in P_i(j)} w_{ui}a_{uj}}{\sum\limits_{u\in P_i(j)} w_{ui}} \tag{2.30}$$

Em casos onde $A_{m\times n}$ é uma matriz esparsa, pode-se construir a matriz completamente especificada $R_{m\times p}$ com posto $p < n$, conforme discutido na seção 2.4.2, subseção "Mitigação do problema de matrizes esparsas". A partir da matriz R calcula-se a vizinhança $P_i(j)$. Uma vez com a vizinhança definida, voltamos à matriz A e aplicamos a equação 2.30 para encontrar a previsão a_{ij} (AGGARWAL, 2016).

2.5.2 Aplicação em sistemas de recomendação

O algoritmo de k-vizinhos mais próximos possui aplicações tanto em sistemas de filtragem baseada em conteúdo como em sistemas de filtragem colaborativa baseados em vizinhança.

Em sistemas baseados em conteúdo o conjunto de dados D compreende uma matriz $X_{m\times n}$, com m observações que apresentam n características cada. A matriz X representa um único usuário u, sendo necessárias r matrizes X para gerar recomendações para r usuários diferentes. Já no caso de sistemas de vizinhança a matriz $X_{m\times n}$ será uma matriz cujos elementos representam as notas dadas por m usuários a n itens.

Enquanto em sistemas baseados em conteúdo a similaridade entre vetores sempre será feita entre itens (similaridade linha a linha), em sistemas baseados em vizinhança é possível encontrar uma previsão $\hat{y}_t$ em termos da similaridade entre usuários (linha a linha) ou entre itens (coluna a coluna). Mais detalhes sobre a implementação de cada método serão dados a seguir.

Exemplo 3. ***K-vizinhos mais próximos em sistemas baseados em conteúdo***

Considere um conjunto de dados D referente a m filmes, com três características cada. Para cada filme temos as informações de gênero, nacionalidade, e número de anos desde a data de lançamento até hoje. Sabemos também se o usuário gostou ou não de cada filme, sendo 1 um indicador de que o usuário gostou do filme e 0 caso contrário.

Queremos saber se o usuário irá gostar ou não de um novo filme, "O Poderoso Chefão III", um drama americano lançado em 1990. A Tabela 6 abaixo exemplifica esse caso.

Código do Filme	Gênero	Nacionalidade	Idade	Avaliação
01	Comédia	Americano	15	1
02	Drama	Alemão	3	0
03	Comédia	Brasileiro	7	0
04	Drama	Brasileiro	9	1
05	Drama	Americano	30	?

Tabela 6 – Avaliações de filmes: Exemplo 3

Após codificar as duas colunas categóricas (gênero e nacionalidade) a matriz X resultante será:

$$X = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 15 \\ 0 & 1 & 0 & 1 & 0 & 3 \\ 1 & 0 & 0 & 0 & 1 & 7 \\ 0 & 1 & 0 & 0 & 1 & 9 \\ 0 & 1 & 1 & 0 & 0 & 30 \end{bmatrix}$$

A fim de evitar viés na construção do modelo iremos excluir as colunas relativas ao gênero drama e à nacionalidade brasileira, e normalizar as colunas.

$$X = \begin{bmatrix} 1 & 1 & 0 & 15 \\ 0 & 0 & 1 & 3 \\ 1 & 0 & 0 & 7 \\ 0 & 0 & 0 & 9 \\ 0 & 1 & 0 & 30 \end{bmatrix}$$

$$X = \begin{bmatrix} 0,707 & 0,707 & 0 & 0,422 \\ 0 & 0 & 1 & 0,084 \\ 0,707 & 0 & 0 & 0,197 \\ 0 & 0 & 0 & 0,253 \\ 0 & 0,707 & 0 & 0,844 \end{bmatrix}$$

Calculamos a distância entre o vetor $\mathbf{x_t}$ *e cada vetor* $\mathbf{x_i}$ *relativo aos filmes já avaliados pelo usuário. Para esse exemplo, tomamos* $d_i = \cos\theta_i$*, sendo* θ_i *o ângulo entre* $\mathbf{x_t}$ *e* $\mathbf{x_i}$*,* $i = 1, ..., 4$*. Os cálculos estão na Tabela 7 abaixo.*

i	$\mathbf{x_t}$	$\mathbf{x_i}$	$\cos\theta_i$
1	(0, 0,707, 0, 0,844)	(0,707, 0,707, 0, 0,422)	0,716
2	(0, 0,707, 0, 0,844)	(0, 0, 1, 0,084)	0,064
3	(0, 0,707, 0, 0,844)	(0,707, 0, 0, 0,197)	0,206
4	(0, 0,707, 0, 0,844)	(0, 0, 0, 0,253)	0,767

Tabela 7 – Similaridade entre filmes: Exemplo 3

Tomando um $k = 2$ *arbritrário, vemos pela Tabela 7 que os dois vetores mais próximos de* $\mathbf{x_t}$ *são os vetores* $\mathbf{x_4}$ *e* $\mathbf{x_1}$*. Vamos calcular o valor esperado de* $\hat{y}_t$ *conforme equação 2.29.*

$$\hat{y}_t = \underset{c_j \in \{0,1\}}{\operatorname{argmax}} \sum_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} F(y_i, c_j)$$

Considerando $c_j = 1$*:*

$$\begin{aligned} \hat{y}_t &= F(y_1, 1) + F(y_4, 1) \\ \hat{y}_t &= F(1, 1) + F(1, 1) \\ \hat{y}_t &= 1 + 1 \\ \hat{y}_t &= 2 \end{aligned} \tag{2.31}$$

Considerando $c_j = 0$*:*

$$\begin{aligned} \hat{y}_t &= F(y_1, 0) + F(y_4, 0) \\ \hat{y}_t &= F(1, 0) + F(1, 0) \\ \hat{y}_t &= 0 + 0 \\ \hat{y}_t &= 0 \end{aligned} \tag{2.32}$$

A condição que maximiza a função 2.29 é quando $c_j = 1$*, então concluímos que o filme "O Poderoso Chefão" é uma boa recomendação para o usuário.*

Exemplo 4. ***K-vizinhos mais próximos em sistemas de vizinhança***

Considere a matriz $M_{m \times n}$ *abaixo cujas entradas* a_{mn} *representam as notas dadas por* m *usuários a* n *itens diferentes,* $m \in \{1, 2, 3, 4, 5\}$ *e* $n \in \{1, 2, 3, 4\}$*. As notas*

são dadas em uma escala de 1 a 5, sendo 1 a pior nota e 5 a melhor. Perceba que alguns dos elementos de M são desconhecidos, e denotados pelo símbolo ?. Esses elementos representam itens que não foram avaliados pelo usuário em questão.

$$M = \begin{bmatrix} 1 & ? & 4 & 5 \\ 3 & 4 & 3 & 4 \\ 2 & 3 & 5 & ? \\ ? & 5 & 3 & 1 \\ 3 & ? & 2 & 4 \end{bmatrix}$$

Vamos calcular o elemento a_{34}, ou seja, a nota que o usuário três daria ao quarto item. Isso pode ser feito de duas formas: usando a similaridade entre usuários ou a similaridade entre itens.

Similaridade entre usuários: *Iniciaremos os cálculos da similaridade entre usuários centralizando os vetores-linha da matriz M, ou seja, subtraindo de cada elemento o valor médio de sua respectiva linha de forma que a média de todas as linhas seja igual a zero. A idéia aqui é diminuir o viés que possa existir quando usuários diferentes avaliam os itens em escalas diferentes. Por exemplo, considere uma escala de avaliações de 1 a 5, sendo 1 a pior nota e 5 a melhor. Pode ser que o usuário i costume dar nota 3 apenas para filmes que ele realmente não gostou. Do contrário, esse usuário costuma dar notas 4 ou 5. Em contrapartida o usuário $i+1$ é mais crítico, e dá nota 3 a itens que gostou, porém não adorou. Para evitar que o modelo seja construído de forma enviezada, centralizamos as notas de cada usuário de forma que todas estejam na mesma escala. É importante notar que a centralização se baseia apenas nos elementos especificados de cada vetor-linha.*

$$M = \begin{bmatrix} -2,3 & ? & 0,7 & 1,7 \\ -0,5 & 0,5 & -0,5 & 0,5 \\ -1,3 & -0,3 & 1,7 & ? \\ ? & 2 & 0 & -2 \\ 0 & ? & -1 & 1 \end{bmatrix}$$

Em seguida, calculamos a similaridade entre usuários. A similaridade é calculada com base apenas nos elementos avaliados por ambos os usuários. Por exemplo, como o primeiro usuário avaliou os itens 1, 3 e 4 e o terceiro usuário avaliou os itens 1, 2 e 3, a similaridade entre esses dois usuários será calculada com base nas notas dadas para os itens 1 e 3 apenas. Da mesma maneira como no exemplo 3 iremos calcular a similaridade entre usuários a partir do cosseno do ângulo entre os vetores correspondentes a cada usuário - quanto mais próximo de 1 o cosseno do ângulo for, mais similares são os usuários. Como queremos calcular a nota dada pelo terceiro usuário ao quarto item, a tabela abaixo apresenta a similaridade entre o terceiro usuário e todos os outros.

Usuário i	$\mathbf{u_3}$	$\mathbf{u_i}$	$\cos\theta_i$
1	(-1,3, 1,7)	(-2,3, 0,7)	0,81
2	(-1,3, -0,3, 1,7)	(-0,5, 0,5, -0,5)	-0,19
4	(-0,3, 1,7)	(2, 0)	-0,17
5	(-1,3, 1,7)	(0, -1)	-0,79

Tabela 8 – Similaridade entre usuários: Exemplo 4

Tomando um $k = 2$ arbitrário, vemos que os usuários mais similares ao usuário 3 e que possuem uma avaliação para o item 4 são os usuários 1 e 4 (caso algum desses dois usuários não tivesse avaliado o item 4, usaríamos o próximo usuário mais similar cuja avaliação para o item 4 é especificada). Para encontrar o valor de a_{34} podemos fazer a média dos valores de a_{14} e a_{44}, ponderada pela similaridade entre os usuários, conforme equação 2.30. A função w_{ui} nesse exemplo é o cosseno do ângulo entre os vetores correspondentes ao usuário u e o usuário i.

$$
\begin{aligned}
a_{ij} &= \frac{\sum_{u \in P_i(j)} w_{ui} a_{uj}}{\sum_{u \in P_i(j)} w_{ui}} \\
a_{34} &= \frac{w_{13} a_{14} + w_{43} a_{44}}{w_{13} + w_{43}} \\
a_{34} &= \frac{(0,81 \times 1,7) + (-0,17 \times -2)}{(0,81 - 0,17)} \\
a_{34} &= 2,68
\end{aligned} \tag{2.33}
$$

Veja que 2,68 é um valor acima de zero (e portanto acima da média) e também acima das notas dadas pelo terceiro usuário a qualquer outro item que ele tenha avaliado, e consequentemente deve ser uma boa recomendação para esse usuário.

***Similaridade entre itens:** Os passos a serem seguidos a fim de calcular a_{34} através da similaridade entre itens são os mesmos passos detalhados anteriormente, com a diferença de que todos devem ser pautados nos vetores-coluna da matriz M, e não em seus vetores-linha. Nesse caso, os vetores serão centralizados com base na média das notas de cada coluna, assim a média de todas as colunas será zero. Além disso, o ângulo entre vetores deverá ser calculado com base nos vetores-coluna da matriz M centralizada. A equação 2.2.19 permanece a mesma, com a diferença de que ao invés de $P_i(j)$ será usado o conjunto $P_j(i)$ com os k itens mais próximos ao item j cujas avaliações para o usuário i são especificadas. O valor de w_{tj} será o cosseno do ângulo entre os vetores correspondentes ao item t e o item j.*

O Algoritmo 2.4 detalha os passos necessários para a implementação do método

dos k-vizinhos mais próximos em um problema de regressão. Mais adiante, o Algoritmo 2.5 aborda as etapas para a utilização do algoritmo em problemas de classificação, mais especificamente em sistemas de recomendação baseados em conteúdo. Os algoritmos 2.6 e 2.7 apresentam a implementação do algoritmo em sistemas de recomendação baseados em vizinhança, abordando a similaridade entre usuários e então a similaridade entre itens.

Algoritmo 2.4 K-vizinhos mais próximos em sistemas baseados em conteúdo: Regressão

Importar base de dados D
Decompor D entre X e $\mathbf{y}$
Codificar colunas em X que possuam dados categóricos
Remover uma coluna de cada variável categórica codificada em X
Normalizar dados em X
Dividir X e $\mathbf{y}$ entre dados de treinamento, dados de validação e dados de teste
Determinar um valor para k
Utilizando os dados de treinamento, calcular a distância entre o vetor $\mathbf{x_t}$, para o qual queremos encontrar uma previsão $\hat{y}_t$, e cada vetor $\mathbf{x_i}$, $i = 1, 2, ..., m$
Encontrar os k vetores $\mathbf{x_i}$ mais próximos ao vetor $\mathbf{x_t}$, definindo a vizinhança $N(\mathbf{x_t}, k)$ de $\mathbf{x_t}$
Determinar a função peso w_i de maneira inversamente proporcional à distância entre vetores
A fim de encontrar $\hat{y}_t$, calcular a média dos valores de y_i ponderada por w_i para todos os k vetores em $N(\mathbf{x_t}, k)$, conforme equação abaixo:

$$\hat{y}_t = \frac{\sum_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} w_i y_i}{\sum_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} w_i}$$

Algoritmo 2.5 K-vizinhos mais próximos em sistemas baseados em conteúdo: Classificação

Importar base de dados D
Decompor D entre X e $\mathbf{y}$
Codificar colunas em X que possuam dados categóricos
Remover uma coluna de cada variável categórica codificada em X
Normalizar dados em X
Dividir X e $\mathbf{y}$ entre dados de treinamento, dados de validação e dados de teste
Determinar um valor para k
Utilizando os dados de treinamento, calcular a distância entre o vetor $\mathbf{x_t}$, para o qual queremos encontrar uma previsão $\hat{y}_t$, e cada vetor $\mathbf{x_i}$, $i = 1, 2, ..., m$
Encontrar os k vetores $\mathbf{x_i}$ mais próximos ao vetor $\mathbf{x_t}$, definindo a vizinhança $N(\mathbf{x_t}, k)$ de $\mathbf{x_t}$
A previsão $\hat{y}_t$ será a classe c_j que irá maximizar a equação:

$$\hat{y}_t = \underset{c_j \in \{c_1, c_2, ..., c_h\}}{\operatorname{argmax}} \sum_{\mathbf{x_i} \in N(\mathbf{x_t}, k)} F(y_i, c_j)$$

Onde $E(y_i, c_j) = 1$ se $y_i = c_j$ e $E(y_i, c_j) = 0$ caso contrário

Algoritmo 2.6 K-vizinhos mais próximos em sistemas de vizinhança: Similaridade entre usuários

Importar matriz de avaliações M
Centralizar vetores-linha de M
Determinar um valor para k
Para encontrar a projeção da avaliação a_{ij} com relação ao usuário i e o item j, calcular a similaridade w_{ui} entre a linha correspondente ao usuário i e todas as outras linhas
Encontrar os k usuários mais similares ao usuário i com avaliações especificadas para o item j, definindo assim o conjunto $P_i(j)$
O valor de a_{ij} será a média dos valores das avaliações ponderada por w_{ui} para todos os k usuários em $P_i(j)$, conforme equação abaixo:

$$a_{ij} = \frac{\sum\limits_{u \in P_i(j)} w_{ui} a_{uj}}{\sum_{u \in P_i(j)} w_{ui}}$$

Algoritmo 2.7 K-vizinhos mais próximos em sistemas de vizinhança: Similaridade entre itens

Importar matriz de avaliações M
Centralizar vetores-coluna de M
Determinar um valor para k
Para encontrar a projeção da avaliação a_{ij} com relação ao usuário i e o item j, calcular a similaridade w_{tj} entre a coluna correspondente ao item j e todas as outras colunas
Encontrar os k itens mais similares ao item j com avaliações especificadas para o usuário i, definindo assim o conjunto $P_j(i)$
O valor de a_{ij} será a média dos valores das avaliações ponderada por w_{tj} para todos os k itens em $P_j(i)$, conforme equação abaixo:

$$a_{ij} = \frac{\sum\limits_{t \in P_j(i)} w_{tj} a_{it}}{\sum_{t \in P_j(i)} w_{tj}}$$

2.6 Árvores de decisão

Algoritmos baseados em árvores de decisão são algoritmos utilizados tanto em problemas de classificação como em problemas de regressão. Sua denominação deve-se à estrutura de gráfico em formato de árvore, com uma raiz, nós e folhas (vide Figura 11 abaixo).

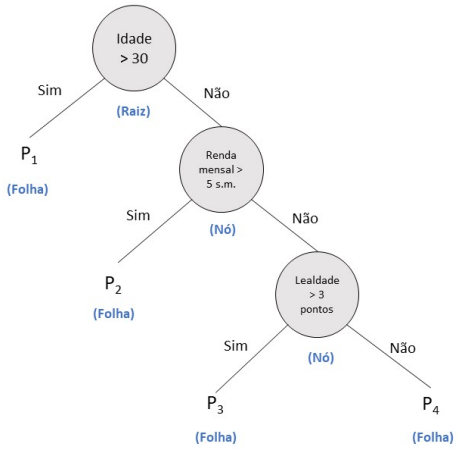

Figura 11 – Exemplo de árvore de decisão cujas folhas apresentam a frequência de compra de um certo produto. Fonte: autora.

O nó de maior hierarquia em uma árvore de decisão é chamado de raiz, sendo representado na figura acima pela pergunta "Idade > 30?". A partir da raiz existem ligações (ramos) para outros nós, chamados de nós internos, ou filhos. Nós que não são conectados a nenhum outro nó são chamados de folhas. O processo de decisão (ou classificação) é realizado com base no valor das folhas. No exemplo acima, o modelo se baseia em várias características de clientes a fim de definir, para um determinado cliente e um dado produto, qual a frequência de compra mais provável. Por exemplo, P_1 pode ser igual a 1 compra por mês, P_2 a 2 compras por mês, P_3 a três compras por mês e P_4 a 4 compras por mês. Com essas informações, uma empresa é capaz de fazer recomendações mais assertivas a seus clientes.

De maneira geral, a construção de uma árvore de decisão envolve os seguintes passos:

1. Escolha das variáveis que serão utilizadas para a definição dos nós, desde a raiz até o último nó;
2. Definição das regras que irão dividir os dados a partir de um determinado nó, com base no valor da variável que define o nó;
3. Definição das regras que definem quando um determinado ramo deve ser encerrado, formando uma folha, ou deve se dividir em novos nós;
4. Definição do valor predito para a variável alvo em cada folha.

No primeiro passo devem ser escolhidas as variáveis mais relevantes para a definição dos nós. Nos algoritmos de árvore de decisão para regressão, isso é feito através

da mensuração da menor distorção quadrática (MSE, do inglês *Minimum Squared Error*), definida por:

$$MSE = \sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{2.34}$$

Onde y_i é o i-ésimo elemento da variável alvo $\mathbf{y}$ e $\hat{y}_i$ a respectiva predição. O MSE dependerá da variável preditiva em questão e de seu respectivo valor de corte, e deverá ser calculado para cada variável considerando um conjunto de valores de corte distintos. O par variável-valor de corte com o menor MSE deve ser escolhido para a definição do próximo nó.

A título de exemplo, considere os dados da Tabela 9, que relacionam a idade e notas atribuídas por leitores para um determinado livro. Suponha que desejamos construir uma árvore de decisão para predizer a nota em função da variável idade.

Idade	**Nota**
12	4
14	5
17	5
18	4
19	4
20	3
23	4
25	3
26	3
29	3
30	2
35	4
38	3
41	1
49	2
52	3
53	1
60	2
65	1

Tabela 9 – Notas para um determinado livro por idade do leitor

Na Figura 12 são apresentadas duas árvores de decisão para esse problema. Na primeira, o nó raiz divide os dados tendo como corte a idade de 31 anos, resultando em um MSE de 17,4. Na segunda, usando a idade de 19,5 anos como corte, o MSE é 14,4. Ou seja, a segunda árvore é melhor do que a primeira.

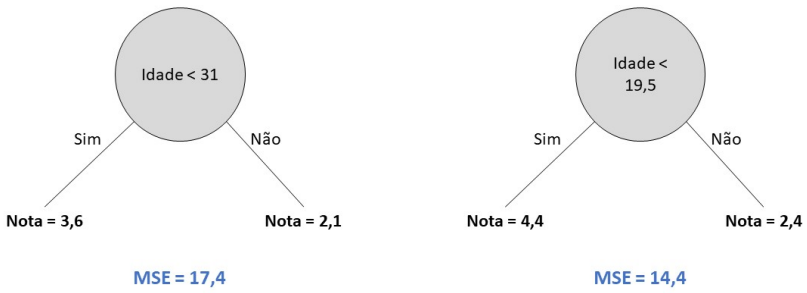

Figura 12 – Exemplo de duas árvores de decisão. Fonte: autora.

Podemos incluir um novo ramo na segunda árvore ao incluir, por exemplo, um novo nó que divide os dados mais uma vez, agora tendo como linha de corte a idade de 38,5 anos. Isso é ilustrado na Figura 13. Nesse caso, o MSE da nova árvore será de 7,4, ou seja, bem melhor que a segunda árvore da Figura 12. Esse processo pode ser repetido iterativamente até atingir o limite do número de ramos. No entanto, nesse processo de ramificação da árvore, deve-se atentar para evitar o *overfitting*.

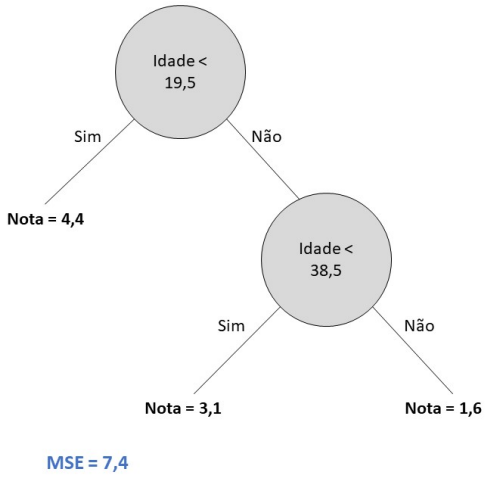

Figura 13 – Exemplo de árvore de decisão com dois níveis. Fonte: autora.

Cada árvore ilustrada na Figura 12 divide os dados em duas regiões, R_1 e R_2, e a predição em cada região pode ser dada, por exemplo, pela média das notas das observações pertencentes àquela região, conforme ilustrado na Figura 14.

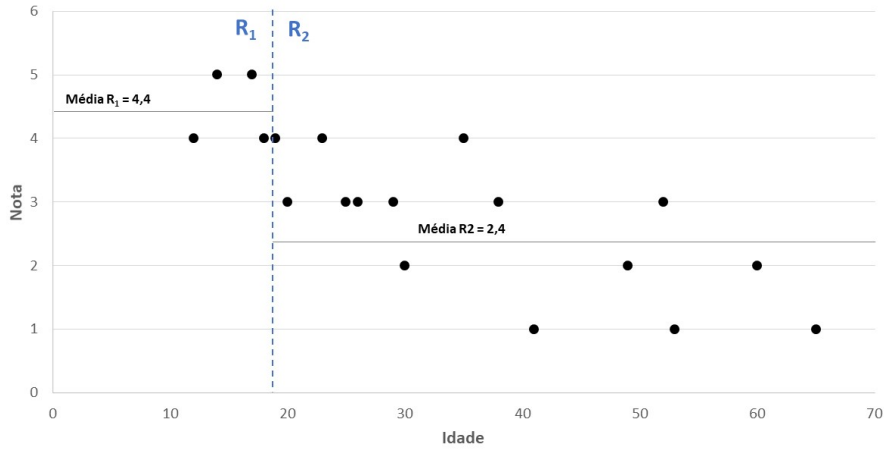

Figura 14 – Ilustração de regiões geradas por uma árvore de decisão. Fonte: autor.

Esse processo pode ser generalizado para considerar várias variáveis e valores de corte, gerando distintas regiões R_u, onde a predição em cada região será dada pela média dos valores da variável alvo correspondentes às observações pertencentes aquela região.

A popularidade dos algoritmos baseados em árvores de decisão se deve a alguns fatores, entre eles a maneira como as soluções são apresentadas de forma a facilitar a visualização e o entendimento. Uma vez tendo a árvore construída, o usuário consegue visualizar exatamente a combinação de variáveis que leva a cada decisão. Do ponto de vista prático as árvores de decisão são aplicáveis tanto a problemas de regressão, cuja variável dependente é numérica, quanto a problemas de classificação, cuja variável dependente é categórica (MARSLAND, 2009).

Entre as dificuldades encontradas na construção e implementação de árvores de decisão está a tendência ao *overfitting*, principalmente quando o número de folhas na árvore é alto. Isso acontece quando o algoritmo "memoriza" os dados previamente conhecidos e perde a capacidade de previsão para novas observaçoes (LAST; MAIMOM; MINKOV, 2002). Além disso, pequenas variações nos dados utilizados na construção do modelo podem gerar árvores muito diferentes entre si (LAST; MAIMOM; MINKOV, 2002). Técnicas de "poda" da árvore, como por exemplo limitar o número de folhas ou de níveis, podem atenuar esses efeitos (MARSLAND, 2009).

2.6.1 Embasamento Matemático

2.6.1.1 Regressão

Existem várias estratégias para se construir uma árvore de decisão (ROKACH; MAIMON, 2015), dentre elas destacaremos, neste trabalho, o algoritmo CART. O método CART, acrônimo para *Classification and Regression Trees*, nos permite construir árvores tanto para dados numéricos, em casos de algoritmos de regressão, como para dados categóricos em algoritmos de classificação (MARSLAND, 2009). Na sequência apresentamos uma breve introdução sobre a construção de uma árvore de regressão utilizando o método CART. Uma referência mais completa pode ser encontrada em (MARSLAND, 2009), (HASTIE; TIBSHIRANI; FRIEDMAN, 2017) e (ROKACH; MAIMON, 2015).

A título de exemplo, considere a árvore representada na Figura 15, gerada a partir de uma base de dados D. Seja D um conjunto de m observações, com n variáveis independentes e uma variável alvo, similar ao ilustrado na Figura 4. Cada observação é definida como um vetor $\mathbf{x_i} \in R^n$, onde $i = 1, ..., m$. O j-ésimo elemento x_{ij} do vetor $\mathbf{x_i}$ corresponde ao valor da variável j para a observação i, sendo $j = 1, ..., n$. A variável alvo é denotada pelo vetor $\mathbf{y} \in R^m$.

A árvore do lado direito da Figura 15 divide os dados em quatro regiões. Para toda observação $\mathbf{x_i}$ em uma região R_u, sendo $u = 1, 2, 3, 4$, o valor $\hat{y}_i$ predito pelo algoritmo será o mesmo. Em outras palavras, para qualquer ponto (x_{i1}, x_{i2}) dentro de R_1, por exemplo, o valor de $\hat{y}_i$ será o mesmo (DOMKE, 2010).

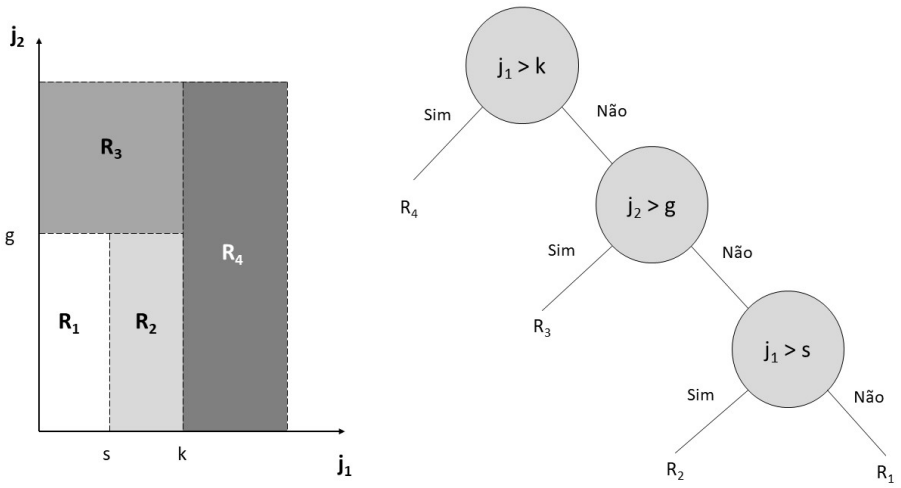

Figura 15 – Árvore de regressão com quatro folhas, representadas pelas regiões R_1, R_2, R_3 e R_4. Fonte: autora.

Como todo $\hat{y}_i$ relativo a um vetor $\mathbf{x_i}$ em uma região R_u tem o mesmo valor, teremos ao fim u valores distintos para $\hat{y}_i$. Para encontrar o valor de $\hat{y}_i$ para cada região u

que minimiza o MSE, ou seja, que minimiza a diferença entre y_i e $\hat{y}_i$, devemos resolver o problema 2.35 abaixo:

$$\hat{y}_i = \underset{\hat{y}_i}{\operatorname{argmin}} \sum_{\mathbf{x_i} \in R_u} (\hat{y}_i - y_i)^2 \tag{2.35}$$

Manipulando o somatório da equação 2.35 teremos:

$$\begin{aligned} S &= \sum_{\mathbf{x_i} \in R_u} (\hat{y}_i - y_i)^2 \\ S &= \sum_{\mathbf{x_i} \in R_u} (\hat{y}_i^2 - 2 y_i y_i + y_i^2) \\ S &= \sum_{\mathbf{x_i} \in R_u} \hat{y}_i^2 - 2\hat{y}_i \sum_{\mathbf{x_i} \in R_u} y_i + \sum_{\mathbf{x_i} \in R_u} y_i^2 \\ S &= N\hat{y}_i^2 - 2\hat{y}_i \sum_{\mathbf{x_i} \in R_u} y_i + \sum_{\mathbf{x_i} \in R_u} y_i^2 \end{aligned} \tag{2.36}$$

Onde N no primeiro termo é o número de vetores $\mathbf{x_i}$ na região R_u. Para minimizar S em 2.36, podemos diferenciar com relação a $\hat{y}_i$:

$$\frac{\partial S}{\partial \hat{y}_i} = 2N\hat{y}_i - 2 \sum_{\mathbf{x_i} \in R_u} y_i \tag{2.37}$$

Igualando a equação 2.37 a zero, concluímos que o valor de $\hat{y}_i$ que minimiza o erro é a média de todos os pontos y_i que correspondem aos vetores $\mathbf{x_i}$ em R_u:

$$\begin{aligned} \frac{\partial S}{\partial \hat{y}_i} &= 0 \\ 2N\hat{y}_i - 2 \sum_{\mathbf{x_i} \in R_u} y_i &= 0 \\ 2N\hat{y}_i &= 2 \sum_{\mathbf{x_i} \in R_u} y_i \\ \hat{y}_i &= \frac{\sum_{\mathbf{x_i} \in R_u} y_i}{N} \end{aligned} \tag{2.38}$$

A partir dos cálculos acima, já estabelecemos que o valor de $\hat{y}_i$ a ser inferido pelo modelo para qualquer $\mathbf{x_i}$ pertencente a uma região R_u será a média de todos os valores y_i relativos a $\mathbf{x_i}$ em R_u (DOMKE, 2010). No entanto, precisamos definir quais serão essas regiões R_u do modelo, e isso implica em definir quais variáveis irão em cada um dos nós, em qual ordem, e qual será o valor de corte que será usado em cada nó para definir os

ramos. Para exemplificar esse processo, considere o exemplo da Figura 15. Veja que a primeira variável definida para ocupar a posição de nó raiz foi a variável j_1. Se j_1 é maior do que k, sendo k o valor de corte da variável j_1 no nó raiz, então o vetor $\mathbf{x_i}$ pertencerá à região R_4. Do contrário, seguimos para investigar o valor de j_2. Se j_2 é maior do que o valor de corte g, então o vetor $\mathbf{x_i}$ pertencerá à R_3. Caso contrário, seguimos para verificar novamente o valor de j_1, porém agora no terceiro nó e não mais no nó raiz. Se j_1 for maior do que a nota de corte s então $\mathbf{x_i}$ pertence à R_2, e se finalmente nenhuma dessas condições foi satisfeita, $\mathbf{x_i}$ pertence à região R_1.

A fim de iniciar os cálculos que estabelecerão as regiões R_u definiremos o vetor $\mathbf{v_j}$ como o vetor-coluna relativo à variável independente v_j, $j = 1, ..., n$, e v_{ij} como o elemento i do vetor $\mathbf{v_j}$. Seja s o valor de corte relativo à variável v_j. Como $s \in R$, existem infinitas possibilidades de escolha para um valor de corte. No entanto, existem apenas alguns valores de s que otimizam a construção da árvore de decisão. Dado um j qualquer, ordenamos os elementos de $\mathbf{v_j}$ em ordem crescente e definimos os valores médios entre cada par $(v_{ij}, v_{(i+1)j})$ (DOMKE, 2010). Ao fim, teremos $m - 1$ possibilidades de s para cada $\mathbf{v_j}$. O cálculo de s é exemplificado abaixo para um valor j fixo:

$$
\begin{aligned}
s'_1 &= \frac{v_{1j} + v_{2j}}{2} \\
s'_2 &= \frac{v_{2j} + v_{3j}}{2} \\
&\vdots \\
s'_{(m-1)} &= \frac{v_{(m-1)j} + v_{mj}}{2}
\end{aligned}
$$

Sendo m o número de observações no conjunto de dados D.

Seguimos a fim de determinar dois conjuntos, T_1 e T_2, tal que:

$$
T_1(j, s) = \{\mathbf{x_i}, x_{ij} \leqslant s\} \qquad T_2(j, s) = \{\mathbf{x_i}, x_{ij} > s\} \tag{2.39}
$$

Os conjuntos T_1 e T_2 dividem os dados da base de dados D em duas regiões. Sejam $\hat{y}_1$ e $\hat{y}_2$ os valores de $\hat{y}_i$ preditos para cada uma dessas duas regiões. Queremos encontrar j e s que irão minimizar o erro entre o valor predito de $\hat{y}_i$ e o valor real y_i:

$$
\underset{j,s}{\operatorname{argmin}} \left[\underset{\hat{y}_1}{\operatorname{argmin}} \sum_{\mathbf{x_i} \in T_1(j,s)} (y_i - \hat{y}_1)^2 + \underset{\hat{y}_2}{\operatorname{argmin}} \sum_{\mathbf{x_i} \in T_2(j,s)} (y_i - \hat{y}_2)^2 \right] \tag{2.40}
$$

Como demonstrado anteriormente, os valores de $\hat{y}_1$ e $\hat{y}_2$ que minimizam os termos internos da equação 2.41 são as médias dos valores de y_i para todo $\mathbf{x_i}$ pertencente

aos conjuntos $T_1(j,s)$ e $T_2(j,s)$. A fim de definir o par (j,s) que irá minimizar o valor da equação 2.41, para todo $j \in (1,2,...,n)$ calcula-se qual o erro entre $\hat{y}_1$ e $\hat{y}_2$ e o valor real y_i. Como $\hat{y}_1$ e $\hat{y}_2$ dependem da escolha de s, esse cálculo deverá ser feito $m-1$ vezes para cada $j \in (1,2,...,n)$. Ao fim da primeira iteração, uma matriz $(m-1) \times n$ poderá ser construída, onde cada valor e_{ij} será o erro relativo à variável v_j com o respectivo valor de corte s'_i. O par (i,j) com o menor valor será escolhido para definir a variável que irá no próximo nó da árvore, assim como o valor de corte (HASTIE; TIBSHIRANI; FRIEDMAN, 2017).

Uma vez com a primeira variável e o valor de corte definidos, repetimos o processo nas duas regiões resultantes da divisão, e assim por diante.

Uma árvore muito grande e que leva em consideração muitas variáveis pode levar ao *overfitting*. Isso acontece porque o algoritmo passa a contar como relevantes variáveis que não necessariamente contribuem para o valor do atributo alvo, tomando casos específicos como regras a serem seguidas. Para evitar esse fenômeno. técnicas de poda podem ser aplicadas. Uma forma fácil de podar a árvore é configurar o processo para que as iterações sejam interrompidas quando o número de observações em uma região chegar a um valor mínimo estabelecido. Existem outros métodos de poda de árvores de decisão, entre eles o *Cost Complexity Criterion*. A idéia é que nós internos de uma árvore sejam retirados e, a cada nó eliminado, um custo é calculado. A sequência de sub-árvores com custo mínimo é a árvore definitiva. Mais detalhes sobre *Cost Complexity Criterion* podem ser encontrados em (HASTIE; TIBSHIRANI; FRIEDMAN, 2017).

2.6.1.2 Classificação

Um dos algoritmos mais populares na construção de árvores de classificação é o algoritmo *ID3*, criado por J. Ross Quinlan (MARSLAND, 2009), e será abordado em mais detalhes nos próximos parágrafos.

A fim de entender o algoritmo *ID3*, é necessário compreender alguns conceitos fundamentais sobre "Teoria da Informação". De maneira simples, a teoria da informação é um campo da matemática aplicada que consiste em estudar a quantidade de informação existente em um sinal (GOODFELLOW; BENGIO; COURVILLE, 2016).

Uma das idéias fundamentais em teoria da informação é a de que saber da ocorrência de um evento improvável nos traz mais informação do que saber da ocorrência de um evento provável. Por exemplo, se alguém disser algo como "isso aconteceu no dia em que o sol nasceu", não é possível captar nenhuma informação que nos ajude a especificar que dia foi esse, uma vez que o sol nasce todos os dias. Agora, se a mesma pessoa nos disser "isso aconteceu no dia em que houve um eclipse", que é um evento muito mais improvável do que o nascer do sol, temos muito mais informação que nos ajude a determinar a data à qual a pessoa se refere.

Seja X uma variável aleatória, e x um dos estados possíveis para essa variável. A informação contida no evento $\mathrm{X} = x$ é definida pela equação 2.41 abaixo:

$$I(x) = -\log P(x) \tag{2.41}$$

Onde $P(x)$ é a probabilidade de que $\mathrm{X} = x$. A unidade de medida de $I(x)$ depende da base do logaritmo utilizado para calcular $I(x)$. Para base e, a unidade de medida é em *nats*. Para base 2, a unidade de medida seria em *bits* ou *shannons*. De forma intuitiva, 1 *nat* seria o total de informação que se ganha ao observar um evento com probabilidade $1/e$.

Enquanto a equação 2.41 nos traz a informação contida em um evento considerando um único resultado (no caso, $\mathrm{X} = x$), é possível calcular a quantidade de informação para uma distribuição de probabilidade completa utilizando o conceito de "Entropia", conforme equação 2.42 abaixo:

$$H(X) = E_{X\sim P}[I(x)] = -E_{X\sim P}[\log P(x)] \tag{2.42}$$

Segundo a equação 2.42, a entropia H de uma variável aleatória X que segue a distribuição P é o valor esperado de informação para um evento proveniente dessa distribuição (GOODFELLOW; BENGIO; COURVILLE, 2016).

A entropia também pode ser vista sob o ponto de vista de incerteza. Para distribuições quase determinísticas, cujo resultado pode ser inferido com quase 100% de certeza, a entropia será baixa (lembre-se do exemplo do nascer do sol: como ocorre todo dia, saber que o sol nasceu quase não nos traz nenhuma informação). Agora imagine uma variável aleatória X cuja distribuição P é uniforme (por exemplo no caso de jogar uma moeda, onde temos 50% de probabilidade de cara e 50% de coroa). Nesse caso a entropia será alta, uma vez que temos um alto nível de incerteza.

O gráfico na figura 16 ilustra a relação entre probabilidade e entropia para uma variável aleatória binária com dois possíveis resultados: x_1 e x_2.

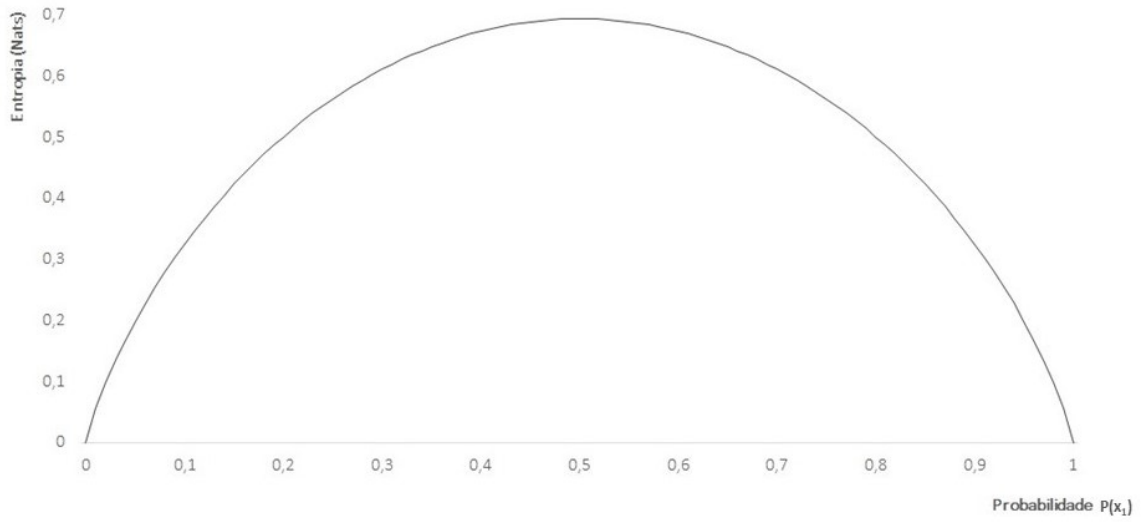

Figura 16 – Entropia para uma variável aleatória binária. Fonte: autora.

Seja o eixo horizontal a probabilidade de x_1. Então, quando $p(x_1) = 0$ temos que $p(x_2) = 1$, e a entropia será:

$$
\begin{aligned}
H(X) &= -E_{X\sim P}[\log P(x)] \\
H(X) &= -\sum_{x\in X} p(x) f(x) \\
H(X) &= -(p(x_1)\log p(x_1) + p(x_2)\log p(x_2)) \\
H(X) &= -(0 + 1\log 1) = 0 \ \textit{Nats}
\end{aligned}
$$

O mesmo pode ser calculado caso $p(x_1) = p(x_2) = 0,5$:

$$
\begin{aligned}
H(X) &= -(p(x_1)\log p(x_1) + p(x_2)\log p(x_2)) \\
H(X) &= -(0,5\log 0,5 + 0,5\log 0,5) = 0,69 \ \textit{Nats}
\end{aligned}
$$

Como ilustrado acima, o pico de entropia para uma variável aleatória binária se dá quando ambos os possíveis resultados tem probabilidade de 50%, ou seja, quando o nível de incerteza é máximo.

De volta à construção de árvores de classificação, utilizamos o conceito de entropia para definir qual será a estrutura da árvore, ou seja, sua raiz, nós, ramos e folhas. O algoritmo *ID3* calcula qual o ganho de informação (quanto maior a entropia da variável maior o ganho de informação) para cada variável independente. A variável que estará no próximo nó é a variável com o maior ganho de informação.

Seja a base de dados D um conjunto de m tuplas $\{\mathbf{x_i}\ ,\ y_i\}$, $i = 1, ..., m$, onde m é o número total de observações, $\mathbf{x_i} \in R^n$, sendo n o número de variáveis de independentes e y_i é uma variável categórica, pertencente a um conjunto de classes $C = \{c_1, c_2, ...c_h\}$, sendo h o número total de classes. Definiremos $\mathbf{v_j}$ como o vetor-coluna relativo à variável independente v_j, $j = 1, 2, ...n$, e x_{ij} como o elemento do vetor $\mathbf{x_i}$ correspondente à variável v_j.

v_1	v_2	v_3	...	v_n	y
x_{11}	x_{12}	x_{13}	...	x_{1n}	c_1
x_{21}	x_{22}	x_{23}	...	x_{2n}	c_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x_{(m-1)1}$	$x_{(m-1)2}$	$x_{(m-1)3}$	...	$x_{(m-1)n}$	c_3
x_{m1}	x_{m2}	x_{m3}	...	x_{mn}	c_h

Tabela 10 – Base de dados D com m observações, n variáveis independentes, e y categórico com h classes

Calculamos o ganho de informação de cada variável independente v_j em D conforme a equação 2.43 abaixo:

$$Ganho(S, j) = Entropia(S) - \sum_{f \in v_j} \frac{|S_f|}{|S|} Entropia(S_f) \tag{2.43}$$

Onde S é o conjunto de todos os vetores $\mathbf{x_i}$ em D, $|S|$ é a contagem de todas as linhas (ou observações) em S, S_f é o subconjunto de S cujas linhas tem valor f para a variável v_j, e $|S_f|$ é a contagem de todas as linhas (ou observações) em S_f.

Para se calcular a entropia de S utilizamos a equação 2.44 abaixo:

$$Entropia(S) = -\sum_{c \in S} p(c) \log p(c) \tag{2.44}$$
$$Entropia(S) = -p(c_1) \log p(c_1) - p(c_2) \log p(c_2) - p(c_3) \log p(c_3) - \ldots - p(c_h) \log p(c_h)$$

Na primeira iteração do algoritmo, $|S|$ será igual a m. Agora tomemos a primeira variável independente, v_1, e calculemos a entropia de cada subconjunto S_f. Voltemos à Tabela 10, porém dessa vez com $m = 4$ e preenchendo com valores a coluna referente a v_1:

v_1	v_2	v_3	...	v_n	y
a_1	x_{12}	x_{13}	...	x_{1n}	c_1
a_3	x_{22}	x_{23}	...	x_{2n}	c_1
a_3	x_{32}	x_{33}	...	x_{3n}	c_3
a_2	x_{42}	x_{43}	...	x_{4n}	c_h

Existem apenas três classes de valores em v_1, $f = a_1, a_2$ e a_3. Com isso, S_{a1}, S_{a_2} e S_{a_3} serão:

S_{a_1}					
v_1	v_2	v_3	...	v_n	y
a_1	x_{12}	x_{13}	...	x_{1n}	c_1

S_{a_2}					
v_1	v_2	v_3	...	v_n	y
a_2	x_{42}	x_{43}	...	x_{4n}	c_h

S_{a_3}					
v_1	v_2	v_3	...	v_n	y
a_3	x_{22}	x_{23}	...	x_{2n}	c_1
a_3	x_{32}	x_{33}	...	x_{3n}	c_3

As entropias de S_{a_1}, S_{a_2} e S_{a_3} podem ser calculadas da mesma forma como foi calculada a entropia de S (equação 2.44), a única diferença é que ao invés de utilizar todo o conjunto S, as entropias agora serão calculadas a partir de subconjuntos de S. O número de observações em cada subconjunto S_f será o valor de $|S_f|$.

Finalmente, o ganho de informação em S da variável v_1 será:

$$Ganho(S, j) = Entropia(S) - \sum_{f \in v_n} \frac{|S_f|}{|S|} Entropia(S_f) \tag{2.45}$$

Substituindo os subconjuntos do exemplo na equação 2.45:

$$Ganho(S, 1) = Entropia(S) - (\frac{|S_{a_1}|}{|S|} Entropia(S_{a_1}) + \frac{|S_{a_2}|}{|S|} Entropia(S_{a_2}) + \frac{|S_{a_3}|}{|S|} Entropia(S_{a_3}))$$

Na primeira iteração do algoritmo, o ganho é calculado conforme a equação acima para todas as n variáveis v_j. A variável com o maior ganho será a variável escolhida para o nó raiz.

Estarão ligados ao nó raiz um número r de ramos, de acordo com o número total de classes existentes na variável definida para representar o nó. Por exemplo, se o maior ganho fosse comprovado ser o ganho referente a v_1, então teríamos três ramos a partir do nó raiz: c_1, c_2 e c_3. Isolando cada uma dessas três classes na base de dados,

obtemos os três subconjuntos já ilustrados anteriormente: S_{a_1}, S_{a_2} e S_{a_3}. A iteração é feita mais uma vez para cada ramo, calculando os ganhos relativos a cada subconjunto para cada variável independente.

Existem vários critérios para determinar o interrompimento desse processo iterativo, como número mínimo de elementos em cada folha, profundidade da árvore, ou quando existe apenas uma classe em $\mathbf{y}$ em um subconjunto S_f (e quando esse é o caso adiciona-se uma folha representando essa classe) (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.6.1.3 Sistemas de filtragem colaborativa

A fim de trabalhar com árvores de decisão em sistemas de filtragem colaborativa, nos deparamos com dois desafios. O primeiro é delimitar quais são as variáveis preditivas e qual é a variável de saída, uma vez que em uma matriz de avaliações $A_{m\times n}$ as variáveis de previsão e de saída não estão claramente definidas como em modelos baseados em conteúdo. O segundo é o problema de matrizes esparsas. Uma vez que um grande número de usuários não terá avaliações conhecidas para vários itens, onde encaixá-los em uma potencial bifurcação na árvore?

Podemos tratar o primeiro problema se definirmos como variável de resposta um item por vez e construirmos n árvores. Por exemplo, se queremos prever qual será a avaliação de um usuário para o item 1, correspondente à primeira coluna em $A_{m\times n}$, construimos a árvore de decisão usando todas as colunas da matriz A como variáveis de previsão, com exceção da primeira coluna, que será tratada como variável de resposta. Ao fim teremos n árvores diferentes. Essas árvores serão utilizadas conforme o item para o qual queremos prever uma avaliação.

O segundo problema pode ser tratado utilizando o método de mitigação do problema de matrizes esparsas através da redução de dimensionalidade da matriz A, método esse discutido na seção 2.4.2. Novamente, se queremos prever qual será a avaliação de um usuário para o item 1, tomamos todas as outras colunas da matriz A, que agora será uma matriz $m \times (n-1)$ e encontramos a matriz completamente especificada $R_{m\times p}$, $p < (n-1)$. Tomamos as p colunas de R como sendo as variáveis de previsão, e a coluna relativa ao item 1 como a variável de saída. Construimos a árvore utilizando, a partir desse novo conjunto de dados, as linhas relativas aos usuários cujas avaliações para o item 1 são conhecidas. Ao final teremos n árvores, cada uma relativa a um item. A fim de fazermos previsões para um usuário u com relação ao item i, utilizamos a árvore construída para o item i a partir das matrizes completamente especificadas (AGGARWAL, 2016).

2.6.2 Aplicação em sistemas de recomendação

Discutiremos a seguir dois exemplos de regressão e classificação para sistemas baseados em conteúdo.

Exemplo 5. ***Árvore de regressão***

Entre os filmes com as maiores bilheterias da história, cinco foram selecionados e são apresentados na tabela abaixo:

Filme	Ano	Bilheteria (USD Milhões)	Avaliação
Avengers: Endgame	2019	2.797,8	4,2
Avatar	2009	2.789,7	3,9
Titanic	1997	2.187,5	4,9
Star Wars: The Force Awakens	2015	2.068,2	2,5
Jurassic World	2015	1.671,7	3,1

Tabela 11 – Avaliações para cinco filmes entre as maiores bilheterias da história

Queremos construir uma árvore de regressão a partir dos dados na tabela acima. Vamos primeiro analisar a variável v_1, relativa ao ano de lançamento do filme. Ordenamos os filmes do mais antigo ao mais novo, e encontramos os candidatos à ponto de corte s':

Ano	Avaliação	s'	Valor de s'
1997	4,9		
2009	3,9	s'_1	2003
2015	2,5	s'_2	2012
2015	3,1	s'_3	2015
2019	4,2	s'_4	2017

Tabela 12 – Árvore de regressão: Primeira iteração com variável ano

Para $s'_1 = 2003$, teremos os seguintes subconjuntos $T_1(j, s)$ e $T_2(j, s)$:

$T_1(ano, 2003)$	
1997	4,9

$T_2(ano, 2003)$	
2009	3,9
2015	2,5
2015	3,1
2019	4,2

Como $T_1(ano, 2003)$ possui apenas uma observação, $\hat{y}_1 = 4,9$ e portanto $y_1 - \hat{y}_1 = 4,9 - 4,9 = 0$, ou seja, o erro relativo à $T_1(ano, 2003)$ é zero. Já em $T_2(ano, 2003)$,

a média das avaliações será $\hat{y}_2 = 3,43$*, e a soma dos erros quadrados será* $1,79$*. A soma dos erros relativos a* $T_1(ano, 2003)$ *e* $T_2(ano, 2003)$ *será então* $0 + 1,79 = 1,79$*.*

Esse processo é repetido para os outros três candidatos à ponto de corte para a variável de ano. Depois, a mesma coisa é feita para a variável de bilheteria. Ao fim, podemos compilar os erros conforme tabela abaixo:

Erros para cada conjunto (v_j, s')		
s	Ano	Bilheteria
s'_1	1,79	3,05
s'_2	1,99	0,71
s'_3	3,24	3,14
s'_4	3,24	3,24

Como o menor erro é relativo ao ponto de corte s'_2 *da variável bilheteria, sabemos que o nó raiz da árvore será correspondente a essa variável. O ponto de corte* s'_2 *para a variável bilheteria é o valor* $2.127,85$*. Existem três bilheterias maiores que* $2.127,85$ *e que formam o primeiro subconjunto, e duas menores ou iguais a* $2.127,85$*, formando o segundo subconjunto. Seguimos para uma segunda iteração do algoritmo, agora performando os mesmos passos acima, porém nos dois subconjuntos resultantes da primeira iteração. Ao final, a árvore de regressão possuirá a seguinte forma:*

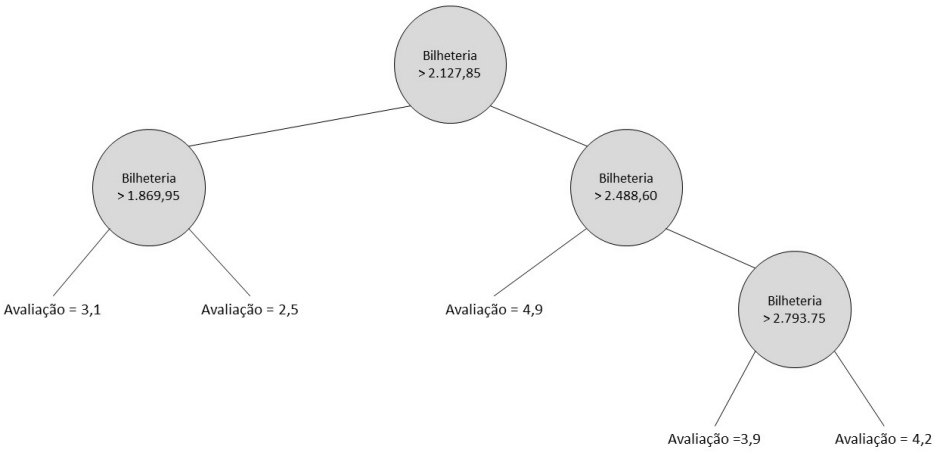

Figura 17 – Árvore de regressão finalizada. Fonte: autora.

Perceba que a árvore apresentada acima poderia ser podada de forma a evitar overfitting. Poderíamos por exemplo limitar a profundidade da árvore a apenas dois níveis, ao invés de três, que é o que a árvore completa possui. Caso a limitação de dois níveis fosse realizada, filmes com bilheteria maior que US$ 2.488.600.000,00 seriam avaliados em

$\frac{3,9+4,9+4,2}{3} = 4,3$, ou seja, a média das avaliações de todos os filmes cuja bilheteria está acima do ponto de corte.

Exemplo 6. ***Árvore de classificação***

Considere a tabela abaixo referente a cinco filmes de diferentes gêneros e distribuidoras. Na coluna de avaliação, o valor 1 corresponde a uma "curtida", ou seja, uma manifestação de interesse por parte do usuário.

Gênero	Distribuidor	Avaliação
Ação	Disney	1
Ação	20th Century	1
Drama	20th Century	0
Ficção	Disney	0
Ficção	Universal	1

Tabela 13 – Avaliações para cinco filmes conforme gênero e distribuidora

Vamos calcular a entropia relativa ao conjunto S, ou seja, o conjunto de dados completo. Temos que $|S|$ é cinco, porque temos cinco observações. Além disso, dentre as cinco observações, três tem avaliação 1 e duas tem avaliação 0, portanto a probabilidade $p(1) = 3/5$ e a probabilidade $p(0) = 2/5$. Calculemos a entropia de S (note que estamos usando log *na base 2, então o valor da entropia se dá em bits).*

$$\begin{aligned} Entropia(S) &= -p(c_1)\log p(c_1) - p(c_2)\log p(c_2) \\ Entropia(S) &= -(3/5)\log(3/5) - (2/5)\log(2/5) \\ Entropia(S) &= 0,97 bits \end{aligned}$$

A fim de definir qual variável irá no nó raiz da árvore, vamos primeiramente encontrar o ganho de informação relativo à variável de gênero. Existem três classes na variável de gênero: ação, drama e ficção. A partir dessas três classes, criamos três subgrupos $S_{\text{ação}}$, S_{drama} e $S_{\text{ficção}}$, conforme abaixo:

$S_{\text{ação}}$		
Gênero	Distribuidora	Avaliação
Ação	Disney	1
Ação	20th Century	1

S_{drama}		
Gênero	Distribuidora	Avaliação
Drama	20th Century	0

$S_{ficção}$		
Gênero	Distribuidora	Avaliação
Ficção	Disney	0
Ficção	Universal	1

É preciso calcular o tamanho e a entropia de cada subgrupo. Para calcular o tamanho, é só contar o número de observações em cada conjunto. Com isso temos que $|S_{ação}|= 2$, $|S_{drama}|= 1$ *e* $|S_{ficção}|= 2$. *Pode-se calcular a entropia da mesma forma como fizemos com o conjunto completo de dados,* S, *utilizando a equação 2.44.*

$$
\begin{aligned}
Entropia(S_{ação}) &= -\sum_{c \in S_{ação}} p(c)\log p(c) \\
Entropia(S_{ação}) &= -p(c_1)\log p(c_1) - p(c_2)\log p(c_2) \\
Entropia(S_{ação}) &= -p(1)\log p(1) - p(0)\log p(0) \\
Entropia(S_{ação}) &= -1\log 1 - 0\log 0 \\
Entropia(S_{ação}) &= 0
\end{aligned}
$$

$$
\begin{aligned}
Entropia(S_{drama}) &= -p(1)\log p(1) - p(0)\log p(0) \\
Entropia(S_{drama}) &= -0\log 0 - 1\log 1 \\
Entropia(S_{drama}) &= 0
\end{aligned}
$$

$$
\begin{aligned}
Entropia(S_{ficção}) &= -p(1)\log p(1) - p(0)\log p(0) \\
Entropia(S_{ficção}) &= -0,5\log 0,5 - 0,5\log 0,5 \\
Entropia(S_{ficção}) &= 1\ bit
\end{aligned}
$$

Podemos agora calcular o ganho de informação da variável gênero, conforme equação 2.43.

$$
\begin{aligned}
Ganho(S, gênero) &= Entropia(S) - \sum_{f \in v_j} \frac{|S_f|}{|S|} Entropia(S_f) \qquad (2.46) \\
Ganho(S, gênero) &= 0,97 - (\frac{2}{5} \times 0 + \frac{1}{5} \times 0 + \frac{2}{5} \times 1) \\
Ganho(S, gênero) &= 0,57
\end{aligned}
$$

O mesmo processo é repetido para a variável distribuidora, onde encontramos que o Ganho(S, distribuidora) é 0,17, que é menor do que o ganho da variável gênero.

Escolhemos portanto a variável gênero para o nó raiz, que terá três ramos: ação, drama e ficção. A partir desses três ramos seguimos com uma segunda iteração, agora em cada um dos subconjuntos $S_{ação}$, S_{drama} e $S_{ficção}$. Caso o subconjunto de dados referente a uma certa classe apresente apenas observações com a mesma avaliação, uma folha será adicionada com o valor dessa avaliação e a segunda iteração para essa classe estará completa. Caso contrário, como para esse exemplo o conjunto de dados S possui apenas duas variáveis independentes, já fica claro que a variável dos nós do segundo nível da árvore será a distribuidora, uma vez que na primeira iteração já foram criados subconjuntos cuja variável gênero é fixa em cada um.

Ao fim do processo, a árvore de classificação será conforme abaixo:

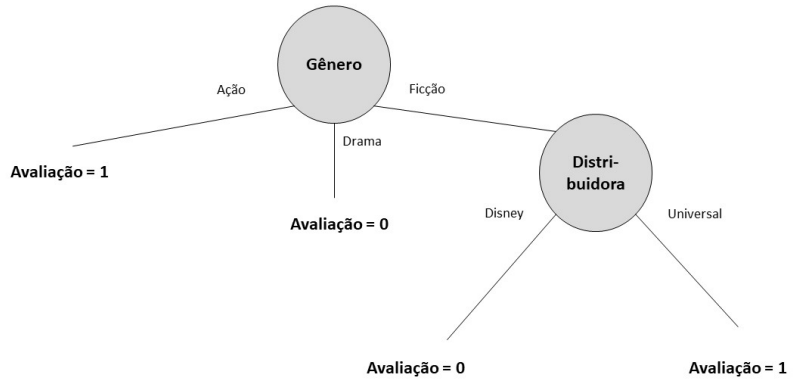

Figura 18 – Árvore de classificação finalizada. Fonte: autora.

São apresentados a seguir dois algoritmos que resumem os principais passos para a construção de árvores de regressão e classificação.

Algoritmo 2.8 Árvore de regressão

Importar base de dados D
Decompor D entre X e $\mathbf{y}$
Codificar colunas em X que possuam dados categóricos
Remover uma coluna de cada variável categórica codificada em X
Normalizar dados em X
Dividir X e $\mathbf{y}$ entre dados de treinamento, dados de validação e dados de teste
Utilizando os dados de treinamento, definir a primeira variável a ser testada para o nó raiz como v_1
Ordenar o conjunto de dados de treinamento de maneira crescente conforme os valores em v_1
Encontrar os $m-1$ candidatos à ponto de corte s'
Definir os primeiros conjuntos T_1 e T_2 de acordo com o primeiro s'
Para cada conjunto definido acima, calcular y_u, $u = 1, 2$, conforme equação abaixo:

$$y_u = \frac{\sum_{\mathbf{x_i} \in T_u} y_i}{N}$$

Encontrar a soma dos erros quadrados entre $\hat{y}_u$ e y_i para T_1 e T_2
Repetir o processo acima para todos os outros candidatos à ponto de corte s', e depois repetir o mesmo processo para as outras variáveis v_j
Definir a variável e o valor de s' com o menor erro como variável e ponto de corte a serem utilizados no nó da árvore
Realizar esse processo iterativamente para cada subdivisão resultante da última definição de variável nó e ponto de corte

Algoritmo 2.9 Árvore de classificação

Importar base de dados D
Decompor D entre X e $\mathbf{y}$
Codificar colunas em X que possuam dados categóricos
Remover uma coluna de cada variável categórica codificada em X
Normalizar dados em X
Dividir X e $\mathbf{y}$ entre dados de treinamento, dados de validação e dados de teste
Utilizando os dados de treinamento, calcular a entropia total do conjunto de dados através da equação abaixo:

$$Entropia(S) = -\sum_{c \in S} p(c) \log p(c)$$

Definir a primeira variável a ser testada para o nó raiz como v_1
Calcular as entropias S_f de cada subgrupo relativo aos valores da variável v_1
Calcular o ganho de informação da variável v_1 através da equação abaixo:

$$Ganho(S, j) = Entropia(S) - \sum_{f \in v_j} \frac{|S_f|}{|S|} Entropia(S_f)$$

Repetir esse processo para as outras variáveis, e definir como variável do nó raiz a que possui o maior ganho de informação
Realizar esse processo iterativamente para cada subdivisão resultante da última definição de variável nó

2.7 Redes Neurais Artificiais

As redes neurais artificiais tem marcado presença em vários dos desenvolvimentos mais modernos e empolgantes de inteligência artificial. Por sua alta versatilidade, ficaram conhecidas como "aproximadoras universais de funções"(AGGARWAL, 2016), podendo ser aplicadas tanto em problemas de classificação como regressão.

Apesar da versatilidade, o maior “trunfo” das redes neurais está na forma como aprendem, simulando a maneira como o próprio cérebro humano adquire conhecimento. Através do fortalecimento ou enfraquecimento dos sinais sinápticos entre neurônios, o cérebro é capaz de consolidar ou não o nosso entendimento a partir de um estímulo externo. Da mesma forma, o aprendizado de algoritmos baseados em redes neurais é feito através do ajuste dos pesos entre os neurônios de uma rede.

Os neurônios (também chamados nós) de uma rede neural são organizados em camadas que podem ser de três tipos: camada de entrada, camadas ocultas e camada de saída. Os nós de cada camada são ligados aos nós da próxima camada através de conexões, sendo que cada conexão possui um peso, conforme ilustrado na Figura 19.

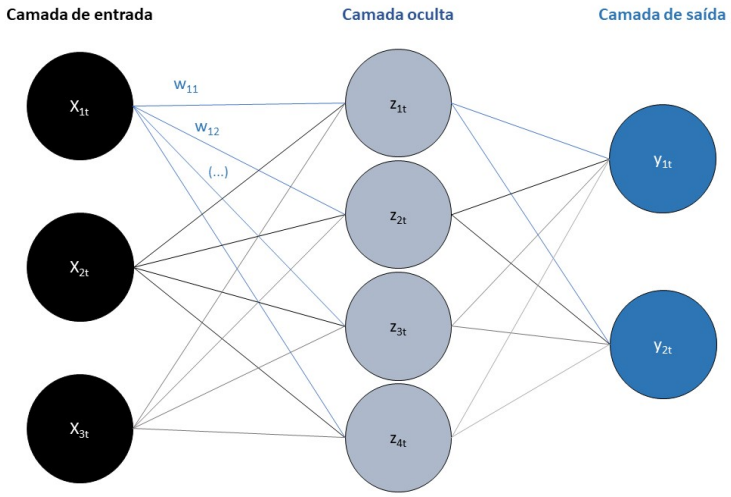

Figura 19 – Figura esquemática de uma rede neural. Fonte: autora.

Denominamos I a quantidade total de nós x_i na camada de entrada, J o número total de nós z_j na camada oculta, e K o número total de nós y_k na camada de saída. Na figura acima o esquema foi desenhado considerando a observação t de um total de T observações. Cada conexão terá peso w_{mn}, onde m representa o índice do nó inicial e n o índice do nó final (STONE, 2020).

O valor total de entrada em um nó é denominado u e também é chamado de *input* total. O *input* u é a soma dos produtos dos valores dos nós da camada anterior e seus respectivos pesos. Designamos a cada nó uma função de ativação $f(u)$, de forma que a saída desse nó (ou *output*) é igual ao resultado da aplicação $f(u)$ no *input* total u.

De maneira genérica, uma rede neural é treinada através do fornecimento de *inputs* aos nós na camada de entrada. Esses *inputs* alimentarão uma função de ativação $f(u)$, e o resultado dessa aplicação será o *output* desses nós. Para cada nó na próxima camada, somamos os produtos entre o *output* dos nós da camada de entrada e os pesos das conexões entre esses nós e o nó da camada atual. Essa soma de produtos será o *input* total dos nós da camada oculta, que por sua vez alimentará uma nova função de ativação que gerará o *output* desses nós. Esse processo é realizado em todos os nós de todas as camadas até chegar nos nós da camada de saída. O erro entre os *outputs* dos nós da camada de saída e o respectivo valor real é calculado, e o valor desse erro é utilizado para atualizar os pesos da rede. Esse processo é feito iterativamente até que o erro esteja dentro de um valor desejado.

2.7.1 Embasamento matématico

Como mencionado anteriormente, o valor do erro calculado ao fim de uma iteração é utilizado para atualizar os pesos de toda a rede. O objetivo final sempre é

minimizar o erro entre o valor de um *output* calculado pela rede neural e seu valor real.

Apesar da existência de dezenas de algoritmos de minimização, um algoritmo amplamente utilizado na minimização de funções em problemas de aprendizado de máquina é o gradiente descendente, que será abordado a seguir.

2.7.1.1 Gradiente Descendente

Considere a rede neural apresentada a seguir, com dois nós na camada de entrada, dois nós na camada oculta, um nó na camada de saída e um peso para cada conexão entre nós:

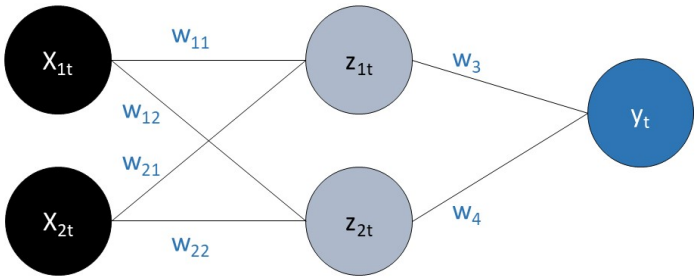

Figura 20 – Rede neural com dois nós na camada de entrada, dois nós na camada oculta e um nó na camada de saída. Fonte: autora.

Seja y_t o valor real do nó de saída para a observação t e $\hat{y}_t$ o *output* calculado pela rede neural. O valor dos *inputs* em z_{1t} e z_{2t} para a observação t serão:

$$\begin{aligned} z_{1t} &= x_{1t}w_{11} + x_{2t}w_{21} \\ z_{2t} &= x_{1t}w_{12} + x_{2t}w_{22} \end{aligned} \tag{2.47}$$

O *output* $\hat{y}_t$ será uma função de z_{1t} e z_{2t}:

$$\hat{y}_t = f(z_{1t})w_3 + f(z_{2t})w_4 \tag{2.48}$$

De maneira didática, consideremos um total de observações $T = 2$. Construimos então duas equações para $\hat{y}_t$, tal que $t \in [1, T]$:

$$\begin{aligned}\hat{y}_1 &= f(z_{11})w_3 + f(z_{21})w_4 \\ \hat{y}_1 &= f(x_{11}w_{11} + x_{21}w_{21})w_3 + f(x_{11}w_{12} + x_{21}w_{22})w_4\end{aligned} \tag{2.49}$$

$$\begin{aligned}\hat{y}_2 &= f(z_{12})w_3 + f(z_{22})w_4 \\ \hat{y}_2 &= f(x_{12}w_{11} + x_{22}w_{21})w_3 + f(x_{12}w_{12} + x_{22}w_{22})w_4\end{aligned} \tag{2.50}$$

O objetivo é encontrar os pesos que minimizem o erro da equação e aproximem y_t e $\hat{y}_t$ ao máximo. Seja $\hat{y}_t = y_t$. Como os valores de x_{1t}, x_{2t} e y_t são conhecidos para cada iteração t, possuímos duas equações e seis variáveis desconhecidas (os pesos w_{11}, w_{12}, w_{21}, w_{22}, w_3 e w_4). Nota-se, com isso, que seria necessário termos várias observações a fim de definir os valores dos pesos e treinar a rede. Considere a inclusão, por exemplo, de mais um neurônio na camada oculta. Criaríamos então três novas conexões e, portanto, três novos pesos a serem treinados.

Essa complexidade se torna ainda mais relevante se considerarmos que as redes neurais aplicadas em problemas reais podem ter alguns milhares (se não milhões) de parâmetros a serem determinados através de treinamento (MOCANU et al., 2018). Esses cenários originam problemas de otimização não linear com muitas variáveis, e podem ser trabalhados com algoritmos iterativos como o método do gradiente descendente.

Para compreender, intuitivamente, como o gradiente descendente por ser utilizado para treinar os pesos de uma rede neural, consideremos $f_{1t} = f(z_{1t})$ e $f_{2t} = f(z_{2t})$. Definimos o vetor $\mathbf{f_t}$ como o vetor com os *outputs* dos nós da camada oculta para cada observação t, tal que $\mathbf{f_1} = (f_{11}, f_{21})$ e $\mathbf{f_2} = (f_{12}, f_{22})$. Construimos também o vetor $\mathbf{w_j} = (w_3, w_4)$. O valor de y_t será então o produto interno entre os vetores $\mathbf{f}_t$ e $\mathbf{w_j}$. Podemos reescrever as equações 2.49 e 2.50 da seguinte forma:

$$\hat{y}_1 = \mathbf{f_1} \cdot \mathbf{w_j} \tag{2.51}$$

$$\hat{y}_2 = \mathbf{f_2} \cdot \mathbf{w_j} \tag{2.52}$$

Uma forma padrão de se calcular o erro E_t entre $\hat{y}_t$ e y_t é:

$$E_t = \frac{1}{2}(y_t - \hat{y}_t)^2 \tag{2.53}$$

Como $\hat{y}_t = \mathbf{f_t} \cdot \mathbf{w_j} = f_{1t}w_3 + f_{2t}w_4$, temos que:

$$E_t = \frac{1}{2}(y_t - (f_{1t}w_3 + f_{2t}w_4))^2 \tag{2.54}$$

O erro total da rede neural da Figura 20 será então a soma dos erros E_t para $t \in [1, T]$:

$$E = \frac{1}{2}\sum_{t=1}^{T}(y_t - (f_{1t}w_3 + f_{2t}w_4))^2$$

$$E = \frac{1}{2}\sum_{t=1}^{T}(y_t - \mathbf{f_t} \cdot \mathbf{w_j})^2 \tag{2.55}$$

Os valores de y_t são conhecidos. Os valores de $\mathbf{f_t}$ dependem de x_{1t}, x_{2t} e das variáveis de peso $w_{11}, w_{12}, w_{21}, w_{22}$. Agrupemos todos os pesos da rede em um vetor $\mathbf{w} = (w_{11}, w_{12}, w_{21}, w_{22}, w_3, w_4)$. Como x_{1t} e x_{2t} também são conhecidos, chegamos à conclusão de que o erro total da rede é uma função dos seis pesos da rede, ou seja, em termos do vetor $\mathbf{w}$. A seguir são apresentados dois exemplos de gráficos do erro E em termos dos pesos da rede, o primeiro considerando uma rede com apenas um peso (uma rede neural com um nó de entrada e um nó de saída), e o segundo considerando uma rede com dois pesos. Para redes com mais do que dois pesos o gráfico do erro passa a ter mais do que três dimensões.

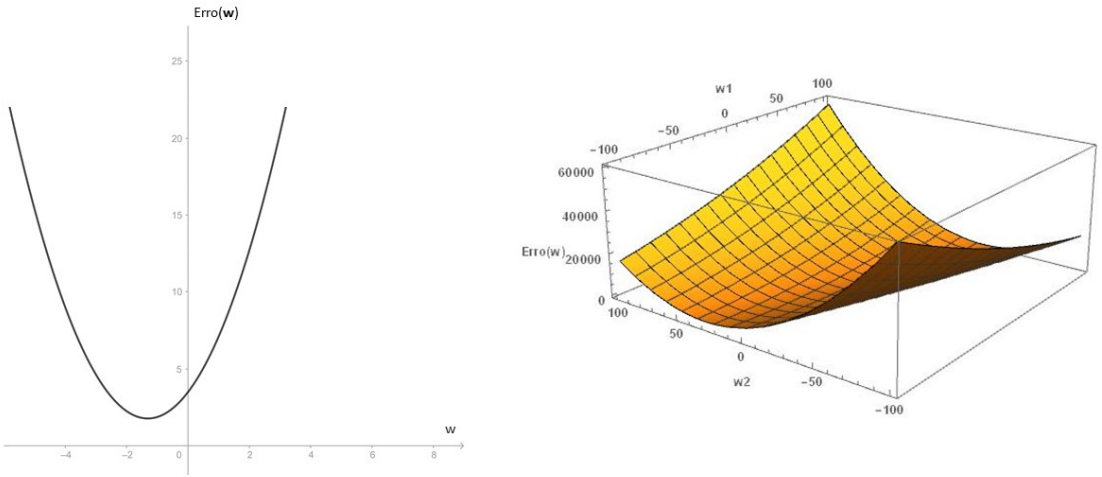

Figura 21 – Função de erro para redes com um peso (esquerda) e dois pesos (direita). Fonte: autora.

Como discutido inicialmente, o objetivo é encontrar o vetor $\mathbf{w}$ que minimize o erro da rede neural. Fazendo uma conexão com a Figura 21 acima, queremos encontrar o valor de $\mathbf{w}$ que nos leve ao ponto mais baixo nos gráficos apresentados, valor esse que aqui chamaremos de $\mathbf{w_{min}}$.

Através do algoritmo do gradiente descendente, iniciamos as iterações com um vetor peso $\mathbf{w_0}$, sendo $\mathbf{w_0}$ um vetor com pesos de valor arbitrário. Calculamos então o vetor gradiente da função erro com relação a $\mathbf{w}$ (ou seja, $\nabla E_{\mathbf{w}}$) no ponto $\mathbf{w_0}$. O vetor gradiente sempre aponta para a direção onde a função cresce mais rápido, e ao chegar no ponto máximo ou mínimo o vetor gradiente possui magnitude zero. Ora, se o gradiente aponta para a direção onde a função cresce mais rápido, o vetor oposto ao vetor gradiente aponta para a direção onde a função diminui mais rápido, o que poderá nos ajudar a minimizar o erro.

Considere o gráfico à esquerda da Figura 21. Se tomamos um valor de $\mathbf{w}$ arbitrário tal que $w = -4$, o vetor oposto ao vetor gradiente nesse ponto apontará para a direção do valor de $\mathbf{w}$ onde a função $E(\mathbf{w})$ é minima, conforme figura abaixo:

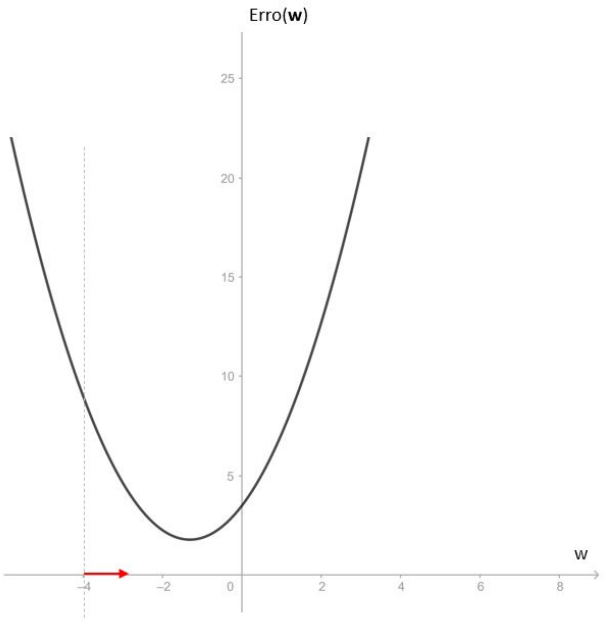

Figura 22 – Função erro e vetor gradiente para w = -4. Fonte: autora.

O algoritmo então atualizará o valor de $\mathbf{w}$ de tal forma que o vetor $\mathbf{w}$ da próxima iteração esteja um pouco mais próximo de $\mathbf{w_{min}}$, conforme equação abaixo:

$$\mathbf{w_{novo}} = \mathbf{w_{velho}} - \epsilon \nabla E_{\mathbf{w}}, \tag{2.56}$$

onde ϵ representa a taxa de aprendizado, ou seja, o tamanho do passo dado em cada iteração na direção oposta ao gradiente. Veja que a diferença entre $\mathbf{w_{novo}}$ e $\mathbf{w_{velho}}$ depende diretamente do valor de $\nabla E_{\mathbf{w}}$. Quando o peso $\mathbf{w}$ ainda está muito longe de $\mathbf{w_{min}}$, a magnitude de $\nabla E_{\mathbf{w}}$ é maior, e damos passos maiores em direção ao ponto mínimo de $E(\mathbf{w})$. Quando já estamos próximos a $\mathbf{w_{min}}$, a magnitude de $\nabla E_{\mathbf{w}}$ é menor, e com isso a diferença entre $\mathbf{w_{novo}}$ e $\mathbf{w_{velho}}$ também é menor.

Se a taxa de aprendizado é muito pequena, quase não há diferença entre $\mathbf{w_{novo}}$ e $\mathbf{w_{velho}}$, e o algoritmo precisa de muitas iterações para chegar ao ponto mínimo. No entanto, se a taxa de aprendizado é muito alta, podemos ir de um ponto a outro entre duas iterações e "pular" $\mathbf{w_{min}}$.

As atualizações no vetor $\mathbf{w}$ são feitas até a convergência, ou seja, até que o erro total da rede atinja um valor assintóticamente ótimo.

Dependendo da natureza da função de erro, o algoritmo do gradiente descendente pode nos levar ao ponto $\mathbf{w}$ onde o erro atinge um mínimo local, e não o mínimo global da função. Veja por exemplo o caso abaixo:

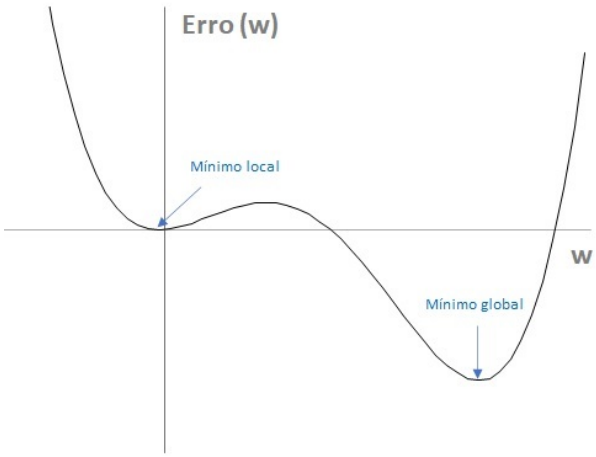

Figura 23 – Função com um mínimo local e um mínimo global. Fonte: autora.

A boa notícia é que redes neurais raramente sofrem do problema de mínimos locais, uma vez que suas superfícies tendem a possuir muitos mínimos locais com a mesma profundidade, e que por sua vez são quase tão profundos quanto o mínimo global (STONE, 2020). Isso fortalece ainda mais o uso do gradiente descendente na construção de redes neurais.

2.7.1.2 Perceptron

Existem inúmeras arquiteturas de redes neurais que se utilizam de várias combinações de nós e camadas. A mais simples é o *Perceptron*, modelo desenvolvido nas décadas de 1950 e 1960.

A arquitetura básica do *Perceptron* consiste em vários nós de entrada e apenas um nó de saída. O nó de saída possui algo que chamamos de *threshold*, que é um limite que determina o valor do *output* desse nó. A função de ativação do nó de saída no caso de um *Perceptron* é uma função degrau unitário, conforme demonstrado na equação 2.57. Caso o *input* total u seja maior do que o *threshold* θ, o *output* do nó de saída sera igual a 1. Caso contrário, será zero.

$$f(u) = \begin{cases} 1, \, u > \theta \\ 0, \, u \leqslant \theta \end{cases} \tag{2.57}$$

Ao adicionarmos camadas ocultas ao *Perceptron*, construimos o que é chamado de *Multilayer Perceptron* (MLP). O problema com um MLP é que, para treiná-lo através do gradiente descendente, é preciso calcular a derivada da função degrau de forma a encontrar o gradiente da função erro. Como a função degrau possui uma descontinuidade quando $u = \theta$, não existe derivada nesse ponto. Por esse motivo, precisamos utilizar uma função contínua e diferenciável. Existem várias opções de funções de ativação para os nós de uma rede neural, e entre as mais comuns está a função sigmoide, que é similar a função degrau unitário porém sem a descontinuidade:

$$f(x) = \frac{L}{1 + e^{-kx}} \tag{2.58}$$

Onde L é o valor máximo da curva e o argumento $-kx$ dita a inclinação da transição. Como estamos procurando uma função similar à função degrau unitário, a função sigmoide $f(u)$ utilizada será:

$$f(u) = \frac{1}{1 + e^{-u}} \tag{2.59}$$

A derivada da função sigmoide com relação ao *input* total u será $\frac{df}{du} = f(u)(1 - f(u))$. Na figura abaixo apresentamos do lado esquerdo a função degrau unitário, e do lado direito a função sigmoide.

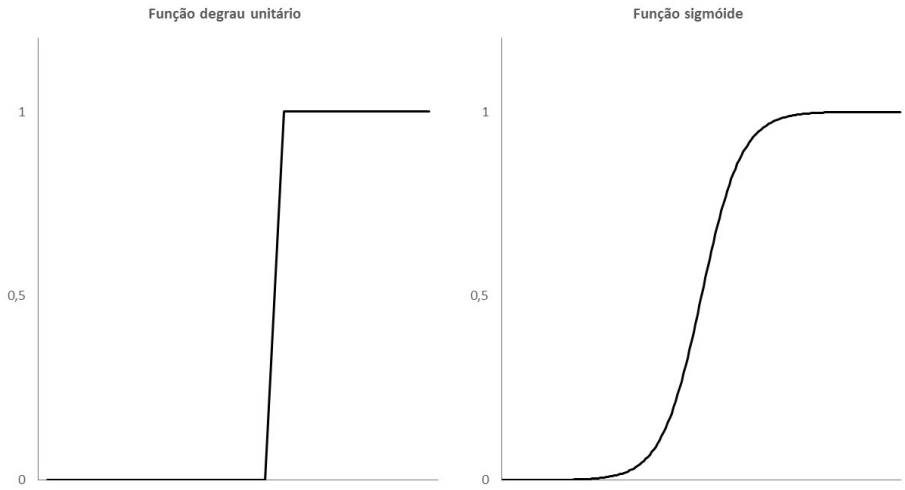

Figura 24 – Função degrau unitário versus função sigmoide. Fonte: autora.

A função sigmoide é especialmente importante por flexibilizar os tipos de *outputs* que a rede neural pode calcular. Se todos os nós possuem funções de ativação lineares, a composição das funções lineares ao longo das várias camadas ainda é uma função linear, ou seja, a adição de camadas pode não acrescentar um valor significativo à rede neural. No entanto, ao adicionar uma camada cuja função ativação é a função sigmoide, conseguimos reproduzir funções não lineares. De fato, uma rede com três camadas de nós sigmoides ou mais é capaz de reproduzir funções que uma rede com apenas duas camadas de unidades sigmoides não consegue (STONE, 2020).

Além disso, não é preciso que os nós nas camadas de entrada e de saída tenham funções de ativação não lineares. O teorema de aproximação universal, provado em sua versão inicial com a função sigmoide por George Cybenko em 1989, afirma que em uma rede neural onde a propagação dos *inputs* acontece da esquerda para a direita (também chamada de *feed-forward* network, que será discutida mais profundamente nos próximos parágrafos) apenas uma camada oculta com um número finito, mas suficientemente grande, de nós cuja função de ativação é não linear é suficiente para aproximar qualquer função contínua em um subconjunto compacto em R^n (STONE, 2020).

2.7.1.3 Algoritmo de *Backpropagation*

Aplicaremos os conceitos do algoritmo de gradiente descendente, a arquitetura *Perceptron* e *Multilayer Perceptron* e a função sigmoide ao que chamamos de algoritmo de *Backpropagation*, que é efetivamente a forma pela qual uma rede neural "aprende".

Considere a rede neural da Figura 25. Nessa rede neural temos $I = 2$, $J = 3$ e $K = 1$. Os *inputs* dos nós x_{it} na camada de entrada são u_i, e da mesma forma os *inputs* da camada oculta são u_j e da camada de saída u_k. Vamos definir as funções de ativação

dos nós das camadas de entrada e de saída como sendo a função identidade, ou seja, $f(u) = u$. A função de ativação na camada oculta será a função sigmoide, de forma que $f(u_j) = (1 + e^{-u_j})^{-1}$.

Veja que adicionamos dois nós nas camadas de entrada e oculta cujo valor é -1. Esses nós são os "vieses" da rede neural, e têm um papel muito parecido com o papel da constante em uma regressão linear, auxiliando o modelo a se ajustar ao conjunto de dados. Aqui, consideramos que o *input* desses dois nós é o valor -1, e o peso entre esses nós e a camada seguinte é designado b_r, onde $r = j$ para o nó na camada de entrada e $r = k$ para o nó na camada oculta.

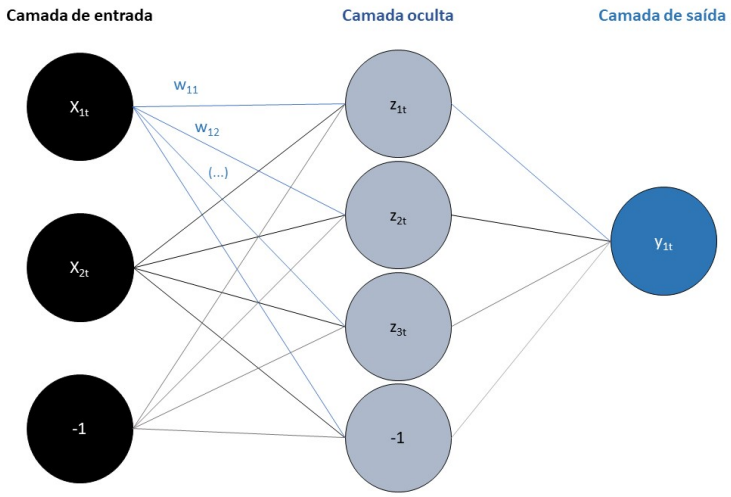

Figura 25 – Rede Neural com vieses. Fonte: autora.

Os *inputs* serão propagados na rede neural da esquerda para a direita, ou seja, indo para a frente, ao que chamamos de *"forward propagation"*. Esse movimento configura uma rede do tipo *feed-forward*, ou na tradução literal, uma rede de alimentação direta.

Para analisar o *forward propagation*, tomemos uma observação t. O *input* total dos nós da camada de entrada serão u_{it}, e como a função de ativação desses nós é linear, temos que o *output* $x_{it} = u_{it}$.

Uma vez que a alimentação da rede é feita da esquerda para a direita, sabemos que o *input* total nos nós na camada oculta será a soma dos produtos dos *outputs* da camada de entrada e seus respectivos pesos, conforme equação 2.60:

$$u_{jt} = \sum_{i=1}^{I} w_{ij} x_{it} - b_j \tag{2.60}$$

Definindo $x_{I+1} = -1$ e $w_{I+1,j} = b_j$, simplificamos a equação:

$$u_{jt} = \sum_{i=1}^{I+1} w_{ij} x_{it} \tag{2.61}$$

Seguindo a mesma lógica, o *input* total no nós da camada de saída será:

$$u_{kt} = \sum_{j=1}^{J+1} w_{jk} z_{jt} \tag{2.62}$$

Sabemos que $z_{jt} = f_j(u_{jt})$ e $y_{kt} = f_k(u_{kt})$. Construimos então a equação de y_{kt} conforme abaixo:

$$\begin{aligned} y_{kt} &= f_k(u_{kt}) \\ y_{kt} &= f_k(\sum_{j=1}^{J+1} w_{jk} z_{jt}) \\ y_{kt} &= f_k(\sum_{j=1}^{J+1} w_{jk} f_j(u_{jt})) \\ y_{kt} &= f_k(\sum_{j=1}^{J+1} w_{jk} f_j(\sum_{i=1}^{I+1} w_{ij} x_{it})) \end{aligned} \tag{2.63}$$

O valor do *output* do nó de saída depende então da combinação entre os *inputs* totais nos nós de entrada, as funções de ativação de cada nó e de todos os pesos da rede. Isso define o que chamamos de *forward propagation* de *inputs*.

Uma vez com os *inputs* propagados da camada de entrada para a camada de saída (de trás para a frente), vamos atualizar os pesos da rede neural através de um mecanismo que faz o caminho contrário, indo da camada de saída para a camada de entrada, chamado de algoritmo de *"backpropagation"* de erros.

Sabemos que, pelo algoritmo do gradiente descendente, a atualização dos pesos da rede é feita utilizando a equação 2.82:

$$\mathbf{w_{novo}} = \mathbf{w_{velho}} - \epsilon \nabla E_{\mathbf{w}} \tag{2.64}$$

Onde:

$$\nabla E_{\mathbf{w}} = (\frac{\partial E}{\partial w_{11}}, \frac{\partial E}{\partial w_{12}}, \ldots, \frac{\partial E}{\partial w_{J+1,K}}) \tag{2.65}$$

Tomemos um peso w_{jk} entre a camada oculta e a camada de saída. Seja Δw_{jkt} a diferença entre $w_{jk_{velho}}$ e $w_{jk_{novo}}$ para o peso w_{jk} na observação t. Temos então que:

$$\Delta w_{jkt} = -\epsilon \frac{\partial E_t}{\partial w_{jk}} \tag{2.66}$$

Como a função erro depende do valor de $y_k t$, que por sua vez é uma função de u_{kt}, e sendo u_{kt} uma função de w_{jk} e z_{jt}, pela regra da cadeia temos que:

$$\Delta w_{jkt} = -\epsilon \frac{\partial E_t}{\partial u_{kt}} \frac{\partial u_{kt}}{\partial w_{jk}} \tag{2.67}$$

Vamos definir a variação do erro E_t na observação t com relação ao *input* u_{kt} no nó de saída y_k como o termo $\delta_{kt} = \dfrac{\partial E_t}{\partial u_{kt}}$.

Sabemos que $u_{kt} = \sum_{j=1}^{J+1} w_{jk} z_{jt}$, e portanto para um w_{jk} fixo $\dfrac{\partial u_{kt}}{\partial w_{jk}} = z_{jt}$. Com isso, concluímos que:

$$\Delta w_{jkt} = -\epsilon \delta_{kt} z_{jt} \tag{2.68}$$

Ou seja, a atualização dos pesos entre a camada oculta e a camada de saída depende do termo δ_{kt} do respectivo nó na camada de saída, iniciando o movimento de propagação "de trás pra frente", o algoritmo de *backpropagation*.

Por definição, $\delta_{kt} = \dfrac{\partial E_t}{\partial u_{kt}}$. Como já mencionado anteriormente, a função erro depende do valor de $y_k t$, que por sua vez é uma função que depende unicamente do valor de u_{kt}. Podemos então aplicar a regra da cadeia em δ_{kt}:

$$\begin{aligned} \delta_{kt} &= \frac{\partial E_t}{\partial u_{kt}} \\ \delta_{kt} &= \frac{\partial E_t}{\partial y_{kt}} \frac{dy_{kt}}{du_{kt}} \end{aligned} \tag{2.69}$$

Como a função de ativação da camada de saída é a função identidade $y_{kt} = u_{kt}$, temos que $\dfrac{dy_{kt}}{du_{kt}} = 1$. Além disso, $E_t = (1/2)(y_{kt} - \hat{y}_{kt})^2$, e $\dfrac{\partial E_t}{\partial y_{kt}} = (y_{kt} - \hat{y}_{kt})$. Temos então que $\delta_{kt} = (y_{kt} - \hat{y}_{kt}) \times 1 = (y_{kt} - \hat{y}_{kt})$.

Disso, concluímos que:

$$\begin{aligned}\Delta w_{jkt} &= -\epsilon\ \delta_{kt}\ z_{jt} \\ \Delta w_{jkt} &= -\epsilon\ (y_{kt} - \hat{y}_{kt})\ z_{jt}\end{aligned} \tag{2.70}$$

E para todas as observações $t \in [1, T]$:

$$\Delta w_{jk} = -\epsilon\ \sum_{t=1}^{T}(y_{kt} - \hat{y}_{kt})\ z_{jt} \tag{2.71}$$

O que conclui a lógica de atualização dos pesos entre a camada oculta e a camada de saída. Nos falta agora definir o processo de atualização dos pesos entre a camada oculta e a camada de entrada. Seguindo a mesma lógica aplicada anteriormente, tomemos um peso w_{ij} entre a camada de entrada e a camada oculta. Seja Δw_{ijt} a diferença entre $w_{ij_{velho}}$ e $w_{ij_{novo}}$ para o peso w_{ij} na observação t. Temos então que:

$$\Delta w_{ijt} = -\epsilon \frac{\partial E_t}{\partial w_{ij}} \tag{2.72}$$

Sabemos que a função erro depende do valor de y_{kt}. O valor de y_{kt} é uma função de u_{kt}, que por sua vez é a soma dos produtos de z_{jt} e os pesos w_{jk}. Sabemos também que o valor de z_{jt} é a aplicação da função sigmoide no *input* u_{jt}, e que u_{jt} é a soma dos produtos de x_{it} e os pesos w_{ij}. Podemos concluir então que pela regra da cadeia:

$$\begin{aligned}\Delta w_{ijt} &= -\epsilon \frac{\partial E_t}{\partial w_{ij}} \\ \Delta w_{ijt} &= -\epsilon \frac{\partial E_t}{\partial u_{jt}} \frac{\partial u_{jt}}{\partial w_{ij}}\end{aligned} \tag{2.73}$$

Vamos definir a variação do erro E_t na observação t com relação ao *input* u_{jt} no nó da camada oculta z_j como o termo $\delta_{jt} = \dfrac{\partial E_t}{\partial u_{jt}}$. Como $u_{jt} = \displaystyle\sum_{i=1}^{I+1} w_{ij} x_{it}$, para um w_{ij} fixo, $\dfrac{\partial u_{jt}}{\partial w_{ij}} = x_{it}$. Com isso, concluímos que:

$$\Delta w_{ijt} = -\epsilon \delta_{jt} x_{it} \tag{2.74}$$

Provamos novamente que a atualização dos pesos acontece de frente para trás, sendo que a diferença entre os pesos novos e velhos para um peso entre a camada de entrada e a camada oculta depende do termo δ_{jt} do respectivo nó na camada oculta.

Por definição, $\delta_{jt} = \dfrac{\partial E_t}{\partial u_{jt}}$. Aplicando a regra da cadeia temos que:

$$\begin{aligned} \delta_{jt} &= \frac{\partial E_t}{\partial u_{jt}} \\ \delta_{jt} &= \frac{\partial E_t}{\partial z_{jt}} \frac{dz_{jt}}{du_{jt}} \end{aligned} \tag{2.75}$$

A função de ativação da camada oculta é a função sigmoide, então $z_{jt} = f(u_{jt}) = (1 - e^{-u_{jt}})^{-1}$. Com isso, a derivada $\dfrac{dz_{jt}}{du_{jt}} = z_{jt}(1 - z_{jt})$.

Para encontrar $\dfrac{\partial E_t}{\partial z_{jt}}$ é preciso lembrar que o erro E_t depende do resultado de todos os nós y_{kt} da camada de saída, que são uma função dos *inputs* u_{kt}. Além disso, sabemos também que o *output* z_{jt} de cada nó na camada oculta contribui para o resultado de todos os nós da camada de saída. Então:

$$\frac{\partial E_t}{\partial z_{jt}} = \sum_{k=1}^{K} \frac{\partial E_t}{\partial u_{kt}} \frac{\partial u_{kt}}{\partial z_{jt}} \tag{2.76}$$

Como $u_{kt} = \sum_{j=1}^{J+1} w_{jk} z_{jt}$, $\dfrac{\partial u_{kt}}{\partial z_{jt}} = w_{jk}$ para um z_{jt} fixo. Além disso, $\dfrac{\partial E_t}{\partial u_{kt}} = \delta_{kt}$ por definição. Com isso temos que:

$$\frac{\partial E_t}{\partial z_{jt}} = \sum_{k=1}^{K} \delta_{kt} w_{jk} \tag{2.77}$$

Substituindo na equação 2.75:

$$\delta_{jt} = z_{jt}(1 - z_{jt}) \sum_{k=1}^{K} \delta_{kt} w_{jk} \tag{2.78}$$

Finalmente, concluímos que para uma observação t:

$$\Delta w_{ijt} = -\epsilon z_{jt}(1 - z_{jt}) \sum_{k=1}^{K} \delta_{kt} w_{jk} x_{it} \tag{2.79}$$

E para todas as observações $t \in [1, T]$:

$$\Delta w_{ij} = -\epsilon \sum_{t=1}^{T} z_{jt}(1 - z_{jt}) \sum_{k=1}^{K} \delta_{kt} w_{jk} x_{it} \tag{2.80}$$

Concluindo o processo para atualizar todos os pesos da rede.

2.7.2 Aplicação em sistemas de recomendação

Por serem extremamente flexíveis, algoritmos de redes neurais podem ser usados tanto em problemas de filtragem baseada em conteúdo quanto em problemas de filtragem colaborativa baseados em modelos.

Em problemas de filtragem baseada em conteúdo, as redes neurais podem ser construídas para atender problemas de regressão ou de classificação. Nesses casos, a matriz de variáveis de previsão $X_{m\times n}$ terá m observações e n características. Em casos de regressão, o vetor com os elementos de saída $\mathbf{y} \in R^m$ possuirá os valores das avaliações de cada observação. Para problemas de classificação, os elementos do vetor $\mathbf{y}$ serão a classe à qual a observação pertence.

Agora, em problemas de filtragem colaborativa, a matriz $X_{m\times n}$ será uma matriz de avaliações de m usuários a n itens. Como nem todos os usuários deram uma nota a todos os itens, muitos elementos são desconhecidos. Podemos usar a rede neural de forma a completar essa matriz iterativamente, criando n redes onde cada um dos itens é o atributo de saída em cada uma das redes. Por exemplo, a primeira rede neural seria construida de forma que a variável v_1 seria o atributo de saída, e os valores das variáveis v_2 a v_n seriam os *inputs* da camada de entrada. Isso pode ser feito n vezes, com n redes neurais diferentes. Ao fim, os valores desconhecidos da matriz original podem ser substituidos pelos outputs de cada instância nas n redes neurais.

Exemplo 7. ***Redes neurais em sistemas baseados em conteúdo***

A fim de discutir problemas de regressão, vamos retomar o exemplo 1. Nesse exemplo temos quatro filmes e suas informações de gênero e nacionalidade, e ao fim o valor das avaliações dadas pelo usuário para cada um.

Após codificação das variáveis, chegamos à Tabela 5. A matriz de variáveis de previsão e o vetor do atributo de saída são:

$$X = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 3,5 \\ 2,7 \\ 3,8 \\ 4,1 \end{bmatrix}$$

Dadas as características do conjunto de dados, podemos construir uma rede neural em cuja camada de entrada teremos cinco nós com entradas binárias. Podemos

então adicionar uma camada oculta com um número inicial J de nós, e apenas um nó na camada de saída que será o valor da avaliação do usuário dadas as características do filme. Podemos determinar as funções de ativação de cada camada, assim como o número de camadas ocultas e o número de nós em cada uma de forma a diminuir o erro entre os valores calculados de $\hat{\mathbf{y}}$ e os valores reais conhecidos $\mathbf{y}$.

Vamos abordar agora um caso de classificação. Considere o exemplo 3 e os filmes descritos na Tabela 3. Nesse caso, o atributo de saída é binário, indicando se o usuário gostou ou não do filme.

Existem algumas poucas diferenças com relação ao caso anteriormente citado de regressão. A primeira é que o conjunto de dados é diferente, e possui a coluna de idade do filme. Em redes neurais é muito importante normalizar os valores dos inputs antes de alimentá-los à rede neural, facilitando o processo de aprendizado do algoritmo. Após codificação das variáveis de classe e normalização dos inputs, teríamos seis nós na camada de entrada. Continuaríamos com apenas um nó na camada de saída, porém o valor do output $\hat{y}_i$ para a observação i seria um valor entre $\hat{y}_i$ 0 e 1. Para valores de $\hat{y}_i$ maiores que um threshold θ, consideraríamos a avaliação como 1. Do contrário, a avaliação seria 0.

Para um número de classes maior (por exemplo se quiséssemos determinar, dadas as características do usuário, se o usuário é do tipo A, B, C ou D e recomendar itens baseados nessa informação), teríamos um camada de saída com quatro nós, e ao fim da iteração a rede neural nos traria uma probabilidade entre 0 e 1 de que o usuário pertence a cada uma dessas classes. A classe com a maior probabilidade será a classe para a qual designamos o usuário.

Exemplo 8. ***Redes neurais em sistemas de filtragem colaborativa baseados em modelos***

Como já discutido anteriormente, matrizes de avaliação utilizadas em sistemas de filtragem colaborativa tendem a ser esparsas, uma vez que nem todos os usuários deram notas a todos os itens existentes em uma plataforma.

Já discutimos com detalhes como é possível mitigar esse problema através da redução de dimensionalidade, criando uma matriz de menor dimensionalidade completamente especificada. O maior problema aqui é a perda de interpretabilidade causada por essa redução de dimensões, o que dificulta o entendimento dos resultados gerados por esse algoritmo.

Uma outra forma de completar matrizes esparsas é através do uso de redes neurais a fim de completá-las iterativamente (AGGARWAL, 2016). Considere a matriz A do exemplo 2, tal que A é uma matriz $m \times n$ gerada a partir das notas dadas por m usuários a n produtos diferentes. As notas variam de 1 a 5, de forma discreta. Abaixo exemplificamos a matriz A para uma amostra de 5 usuários e 3 produtos.

$$A = \begin{bmatrix} 2 & 2 & ? \\ 4 & 3 & 1 \\ ? & 4 & 5 \\ 2 & 1 & 4 \\ 4 & 5 & 2 \end{bmatrix}$$

A primeira coisa a se fazer é preencher os elementos faltantes com as médias de cada coluna. A fim de tornar o resultado ainda mais robusto, é possível primeiro centralizar cada coluna tal que a média de cada uma é zero, e então os elementos faltantes, que serão a média de cada coluna, também serão zero, conforme matriz abaixo:

$$A = \begin{bmatrix} -1 & -1 & 0 \\ 1 & 0 & -2 \\ 0 & 1 & 2 \\ -1 & -2 & 1 \\ 1 & 2 & -1 \end{bmatrix}$$

Como alternativa, poderíamos também empregar um algoritmo simples como por exemplo k-vizinhos mais próximos a fim de preencher a matriz pela primeira vez.

A partir dessa primeira matriz completa, elegemos a primeira coluna relativa ao primeiro item como o vetor dos atributos de saída, e usamos o restante da matriz, que já está completa, como a matriz das variáveis de previsão. Por exemplo, no caso da matriz A, a matriz X de variáveis de previsão e o vetor y dos atributos de saída seriam:

$$X = \begin{bmatrix} -1 & 0 \\ 0 & -2 \\ 1 & 2 \\ -2 & 1 \\ 2 & -1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} -1 \\ 1 \\ ? \\ -1 \\ 1 \end{bmatrix}$$

Usaríamos então as observações para as quais o respectivo elemento no vetor $\mathbf{y}$ *é conhecido como conjunto de dados de treinamento da rede neural (nesse exemplo seriam as observações das linhas 1, 2, 4 e 5), e as observações para as quais o respectivo elemento*

em $\mathbf{y}$ *não é especificado como as observações para as quais queremos prever o valor de* y *(a linha 3).*

Montamos então a primeira rede neural, com dois nós na camada de entrada, referentes aos valores dos itens nas colunas 2 e 3 da matriz X*, e um nó na camada de saída para a previsão do valor de* $\mathbf{y}$*. Após empregar a rede neural para a previsão do valor de* y*, substituimos o terceiro elemento da primeira coluna de* A *por* y*, reservamos esse resultado, e seguimos a fim de realizar o mesmo processo novamente, agora considerando a terceira coluna de* A *como o vetor de atributos de saída e as colunas 1 e 2 a fim de formar a matriz de variáveis de previsão.*

Esse processo é realizado n *vezes, sendo* n *o número de itens (ou colunas) para as quais existe um elemento não especificado. Após a finalização desse processo, teremos uma nova matriz* $A_{m\times n}$ *cujas entradas não especificadas foram atualizadas. O processo é então realizado uma segunda vez, e é feito até que os valores dos elementos desconhecidos inicialmente convirjam.*

Com a matriz A *completa, podemos decidir quais itens devem ser sugeridos a quais usuários.*

A seguir apresentamos dois algoritmos. O primeiro resume os passos para a construção de uma rede neural aritifical em sistemas baseados em conteúdo. Logo em seguida, apresentamos um algoritmo que detalha a implementação de uma rede neural em sistemas de filtragem colaborativa.

Algoritmo 2.10 Redes neurais em sistemas baseados em conteúdo

Importar matriz de variáveis de previsão X e vetor do atributo de saída $\mathbf{y}$
Normalizar colunas em X
Dividir X e $\mathbf{y}$ entre dados de treinamento, dados de validação e dados de teste
Definir a topologia da rede neural da seguinte forma: camada de entrada com n neurônios, camada de saída com um neurônio para problemas de regressão e c neurônios para problemas de classificação, onde c é o número total de classes do conjunto de dados. Defina t camadas ocultas com s neurônios cada, onde t e s podem ser ajustados a fim de melhorar a performance do algoritmo
Utilizando as observações para as quais o valor em $\mathbf{y}$ é especificado, treinar a rede neural usando o algoritmo do gradiente descendente:

$$\mathbf{w_{novo}} = \mathbf{w_{velho}} - \epsilon \nabla E_{\mathbf{w}} \tag{2.81}$$

Após treino do algoritmo, realizar a previsão dos valores não especificados em $\mathbf{y}$

Algoritmo 2.11 Redes neurais em sistemas de filtragem colaborativa

Importar matriz de avaliações A

Centralizar vetores-coluna de A e preencher elementos faltantes com a média de cada coluna, ou seja, zero

Para a primeira iteração, determinar a primeira coluna de A como o vetor $\mathbf{y}$ com os atributos de saída, e as outras colunas formam a matriz X de variáveis de previsão

Definir a topologia da rede neural da seguinte forma: camada de entrada com n-1 neurônios, camada de saída com um neurônio, e t camadas ocultas com s neurônios cada, onde t e s podem ser ajustados a fim de melhorar a performance do algoritmo

Definir as funções de ativação para os neurônios em cada camada

Utilizando as observações para as quais o valor em $\mathbf{y}$ é especificado, treinar a rede neural usando o algoritmo do gradiente descendente:

$$\mathbf{w_{novo}} = \mathbf{w_{velho}} - \epsilon \nabla E_{\mathbf{w}} \tag{2.82}$$

Após treino do algoritmo, realizar a previsão dos valores não especificados em $\mathbf{y}$

Reserve o vetor $\mathbf{y_{novo}}$ e repita o processo para as $n-1$ colunas faltantes, construindo a nova matriz A_{nova}

Repita os passos acima até que os valores na matriz A convirjam

3 Aplicações

3.1 Apresentação do problema

O objetivo dessa seção é apresentar ao leitor diferentes aplicações de alguns dos algoritmos discutidos no segundo capítulo. Para tal, iremos explorar os dados relativos aos clientes do *e-commerce* americano *ModCloth*, especializado na venda de roupas femininas. O objetivo é criar sistemas de recomendação utilizando diferentes algoritmos porém com o mesmo fim: apresentar ao cliente recomendações de novos itens com a intenção de aumentar as vendas e ampliar a percepção de valor do cliente sobre a loja.

3.1.0.1 Estrutura dos dados

A base de dados utilizada nas aplicações a serem discutidas foi originalmente apresentada por Rishabh Misra, Mengting Wan e Julian McAuley na décima segunda conferência sobre sistemas de recomendações da *Association for Computing Machinery* (MISRA; WAN; MCAULEY, 2018). Em sua estrutura original, possui as avaliações de 47.958 usuários sobre 1.378 itens. As avaliações seguiam notas de 1 a 5. O usuário com o menor número de itens avaliados avaliou apenas uma peça de roupa, enquanto o usuário que avaliou o maior número de itens avaliou 27 peças. No total, a base original possui 82.790 avaliações.

O detalhamento dos campos da base de dados original é apresentado na Tabela 14.

Campo	**Detalhamento**	**Exemplo**
item$_id$	Código numérico representando o item avaliado	125442
waist	Número representando a medida da cintura do usuário	31
size	Número representando a medida geral do usuário	11
quality	Avaliação de 1 a 5, em um intervalo discreto, feita pelo usuário ao item em questão	3,0
cup size	Informação categórica representando a medida do busto do usuário	dd/e
hips	Número representando a medida do quadril do usuário	36
bra size	Número representando outra medida do busto do usuário	34
category	Categoria do item em questão	Dresses
bust	Número representando ainda outra medida do busto do usuário	36
height	Altura do usuário no sistema imperial	5ft 7in
user$_name$	Nome do usuário	biv$_is$
length	Informação categórica sobre comprimento do item	just right
fit	Informação categória sobre tamanho do item	small
user$_id$	Código numérico representando o usuário que fez a avaliação	575252
shoe size	Número representando o tamanho dos pés do usuário	9,0
shoe width	Informação categórica sobre a largura dos pés do usuário	average
review$_summary$	Para as avaliações onde o usuário escreveu algum texto com detalhamentos, o resumo desse texto	"Suits my body type!"
review$_text$	O texto escrito pelo usuário detalhando a avaliação	"From the other reviews it seems like this dress either works for your body type or it doesn't..."

Tabela 14 – Detalhamento de campos da base de dados da ModCloth

A fim de diminuir o tamanho da base de dados para agilizar os testes, além de evitar a criação de um viés nos resultados por conta de itens que tiveram apenas uma ou poucas avaliações, a base original foi filtrada a fim de manter apenas as avaliações referentes a itens que tinham sido avaliados por pelo menos 500 usuários. Além disso, foram removidos alguns campos cuja qualidade dos dados estava muito comprometida

(informações mal escritas ou faltando), e feita a conversão do campo de altura do usuário, com unidades em pés e polegadas, para o sistema métrico em centímetros.

Após esse processo de limpeza e formatação dos dados, a base gerada possui 41.855 avaliações relativas a 50 itens (todos com pelo menos 500 avaliações cada) e 30.174 usuários. Dos campos originais, foram mantidas as colunas de *item_id*, *category*, *user_id*, *height*, *size*, *hips*, *bra size* e *quality*.

Todo o tratamento de dados e implementação dos algoritmos foram realizados utilizando a linguagem de programação Python.

3.2 Algoritmos utilizados

Ao todo foram empregados quatro algoritmos para a criação de dois sistemas de filtragem colaborativa e dois sistemas de filtragem baseada em conteúdo, conforme Tabela 15.

Tipo de sistema de recomendação	**Algoritmo**
Filtragem colaborativa	Método dos k-vizinhos mais próximos
Filtragem colaborativa	Decomposição em valores singulares
Filtragem baseada em conteúdo	Método dos k-vizinhos mais próximos
Filtragem baseada em conteúdo	Redes neurais artificiais

Tabela 15 – Detalhamento dos algoritmos aplicados

O objetivo foi utilizar o mesmo conjunto de dados nos quatro algoritmos, nos permitindo comparar os resultados tanto entre os dois tipos de filtragem como entre os tipos de algoritmos empregados em cada filtragem.

3.2.1 Sistemas de filtragem colaborativa

Para os sistemas de filtragem colaborativa foi utilizado o pacote de ferramentas *Surprise*, um pacote desenvolvido para uso em Python e que facilita muitíssimo a aplicação de algoritmos de filtragem colaborativa na construção de sistemas de recomendação. Em ambos os casos a única variável considerada nas aplicações foi a variável de *quality* para cada combinação usuário-item. Perceba que a matriz usuário-item utilizada nos algoritmos de filtragem colaborativa possui as avaliações de 30.174 usuários sobre 50 itens, dessa forma possuindo $50 \times 30.174 = 1.508.700$ elementos. No entanto, temos apenas 41.855 avaliações especificadas, o que quer dizer que 97,2% dos valores da matriz são desconhecidos, configurando uma matriz esparsa.

3.2.1.1 Método dos k-vizinhos mais próximos

Para gerar as recomendações através do método dos k vizinhos mais próximos, utilizamos o algoritmo *KNNWithMeans* do pacote *Surprise*. Conforme documentação do algoritmo (HUG, 2015), a similaridade é calculada entre usuários de maneira padrão (da mesma forma como foi discutido no exemplo 4 do Capítulo 2). Definimos a função similaridade como sendo o cosseno do ângulo entre os vetores relativos a cada usuário. Além disso, a nota média de cada usuário é utilizada para centralizar os vetores-linha da matriz usuário-item, de maneira a diminuir o possível viés quando usuários diferentes avaliam o mesmo item em escalas diferentes.

A avaliação centralizada de cada usuário para cada item não avaliado é calculada então com base na equação 2.30. A fim de encontrar as avaliações não centralizadas, basta somar a nota média de cada usuário a todas as avaliações centralizadas estimadas pelo algoritmo.

De maneira padrão, o algoritmo *KNNWithMeans* utiliza k = 40, ou seja, são considerados no máximo 40 usuários entre os usuários mais similares ao usuário em questão. Em outras palavras, o número máximo de usuários na vizinhança do usuário u será 40. Lembre-se que, para que um usuário v seja considerado no conjunto de usuários similares ao usuário u a fim de estimar a avaliação do usuário u ao item i, a avaliação do usuário v ao item i precisa ser especificada. Além disso, no momento da geração da vizinhança para o usuário u são considerados apenas usuários cuja similaridade com o usuário u é positiva. Caso não exista nenhum usuário v na vizinhança de u para avaliação ao item i, a soma no numerador da equação 2.30 será zero e a estimativa da avaliação do usuário u será apenas nota média desse usuário, considerando todos os itens já avaliados por ele.

3.2.1.2 Decomposição em valores singulares

Utilizamos o algoritmo *SVD* do pacote *Surprise* para gerar as recomendações utilizando o método da decomposição em valores singulares. A fim de aplicar a decomposição em valores singulares a uma matriz, é preciso que essa matriz esteja completamente especificada, ou seja, as 97,2% avaliações não conhecidas precisariam ser substituídas com algum tipo de valor inicial a fim de iniciarmos o processo de decomposição. Poderíamos realizar o processo de inicialização da matriz substituindo cada elemento desconhecido pela média de avaliações de cada item (conforme trabalhado no exemplo 2), no entanto, dado o tamanho da matriz (30.174×50), trabalhar dessa forma tornaria o algoritmo extremamente pesado e lento.

A fim de mitigar esse problema, o algoritmo *SVD* do pacote *Surprise* trabalha com a criação do modelo utilizando apenas as avaliações especificadas. A fim de determinar os fatores latentes e consequentemente os vetores $\mathbf{u_i}$ e $\mathbf{v_j}$ necessários para estimar a

avaliação a_{ij} de um usuário i a um item j (conforme equação 2.12), é feita a otimização da seguinte equação abaixo (KOREN; BELL; VOLINSKY, 2009):

$$min_{(v*,u*)} = \sum_{i,j \in K} (a_{ij} - \mathbf{v_j}^T \ \mathbf{u_i})^2 + \lambda(|\mathbf{v_j}|^2 + |\mathbf{u_i}|^2) \tag{3.1}$$

Onde $\lambda = 0,02$ e é uma constante responsável pela regularização do modelo. O conjunto K é o conjunto de todos os pares de usuários e itens (i, j) para os quais conhecemos as avaliações a_{ij}. Para o conjunto de dados trabalhado nesse capítulo, o número de elementos em K é 41.855.

Ao invés de calcular as avaliações desconhecidas a_{ij} simplesmente através do produto interno entre $\mathbf{u}_i$ e $\mathbf{v}_j$, adicionamos ao algoritmo um ajuste que diz respeito aos "vieses" implícitos às avaliações conhecidas. Esses vieses são responsáveis por quantificar a tendência de alguns itens serem melhor avaliados do que outros (como por exemplo quando a percepção popular de uma marca é melhor do que de outra) e a tendência de alguns usuários serem mais ou menos críticos em suas avaliações.

Ao final, a avaliação a_{ij} calculada pelo algoritmo seguirá a seguinte equação abaixo:

$$a_{ij} = \mu + b_i + b_j + \mathbf{v_j}^T \ \mathbf{u_i} \tag{3.2}$$

Onde μ é a média geral das avaliações conhecidas, para todos os usuários e todos os itens. A constante b_i é o viés do usuário em questão, e a constante b_j é o viés do item em questão. Por exemplo, suponha que estejamos trabalhando com uma matriz de avaliações normalizada, para a qual a média global μ é zero. Sabemos que o usuário i é mais crítico e tende a dar notas -0,5 pontos mais baixas que a média. Além disso, o item j é bem avaliado pelo mercado, e por isso costuma receber avaliações 1 ponto acima da média. Se o produto interno $\mathbf{v_j}^T \ \mathbf{u_i}$ tem a nota 2,5 como resultado, podemos somar a isso os valores $\mu = 0$, $b_i = -0,5$ e $b_j = 1$, chegando ao resultado final $a_{ij} = 3$.

Substituindo a equação 3.2 na equação 3.1, concluímos que a função a ser otimizada será:

$$min_{(v*,u*)} = \sum_{i,j \in K} (a_{ij} - \mu - b_i - b_j - \mathbf{v_j}^T \ \mathbf{u_i})^2 + \lambda(|\mathbf{v_j}|^2 + |\mathbf{u_i}|^2 + b_i^2 + b_j^2) \tag{3.3}$$

3.2.2 Sistemas de filtragem baseada em conteúdo

Construímos dois sistemas de filtragem baseada em conteúdo. Através das informações conhecidas sobre cada usuário (foram utilizadas as variáveis $height, size, hips$

e *bra size*), foram construídos modelos utilizando o método dos k-vizinhos mais próximos e também redes neurais artificiais.

Como alguns dos valores para as características de alguns usuários estavam faltando, os elementos desconhecidos foram preenchidos com a média dos valores para cada item. Por exemplo, se a altura para certo usuário era desconhecida, preenchemos esse valor com a média das alturas de todos os usuários. Essa técnica nos permite seguir com a criação do modelo sem comprometer o volume de informação que possuímos - se fôssemos excluir da análise cada usuário para o qual falta alguma informação, a base de dados final a ser utilizada pelo modelo seria muito menor do que a original.

3.2.2.1 Método dos k-vizinhos mais próximos

A fim de aplicar o algoritmo dos k-vizinhos mais próximos em um sistema de filtragem baseada em conteúdo, utilizamos o pacote *KNeighborsRegressor* da biblioteca *sklearn*. Definimos o problema como uma regressão, ou seja, assim como nos algoritmos de sistema de filtragem colaborativa discutidos acima, a avaliação estimada a_{ij} do usuário i ao item j será considerada como um número real entre 1 e 5.

A função utilizada para calcular a similaridade entre usuários foi a distância euclidiana entre os vetores correspondentes a cada usuário, sendo que o vetor $\mathbf{c}_i$ referente às características do usuário i possui quatro elementos numéricos, um elemento para cada característica. A base foi normalizada de forma que todas as variáveis estivessem na mesma escala. Assim como no sistema de filtragem colaborativa, utilizamos k = 40.

Foram criados 50 modelos, um para cada um dos itens na base de dados. Entre os 50 itens, foram listados todos os itens j não avaliados por cada um dos 30.174 usuários. Para cada item não avaliado pelo usuário, aplicamos o modelo correspondente a fim de estimar a nota que esse usuário daria ao item em questão (perceba aqui que, enquanto criamos apenas um modelo para gerar todas as avaliações desconhecidas nos sistemas de filtragem colaborativa, foi necessário criar um modelo por item nos sistemas de filtragem baseada em conteúdo).

3.2.2.2 Rede Neural Artificial

Para construir a rede neural artificial utilizada nesse sistema de recomendação, utilizamos a biblioteca para *deep learning* em Python, chamada *Keras*. Assim como em todos os outros algoritmos já discutidos, o modelo foi construído de forma que o *output* seja um número real entre 1 e 5, configurando um problema de regressão.

Após algumas tentativas, a topologia que apresentou os melhores resultados foi definida, sendo que a camada de entrada possui 4 neurônios (um para cada característica de cada usuário), a primeira camada oculta possui 8 neurônios, a segunda camada oculta

possui 20 neurônios, e a camada de saída possui apenas um neurônio, por se tratar de um problema de regressão. A função de ativação dos neurônios em todas as camadas ocultas foi definida como a função sigmóide.

Da mesma forma como no modelo baseado em conteúdo criado a partir dos k-vizinhos mais próximos, os atributos de entrada foram normalizados antes de serem utilizados na rede neural. Além disso, também é necessário criar 50 modelos, um modelo para um dos 50 itens. Para cada item j não avaliado por um usuário i, utilizamos as características desse usuário como *input* no modelo do item correspondente, de maneira que o *output* será um valor entre 1 e 5 relativo à nota a_{ij} estimada da avaliação do usuário i ao item j.

3.3 Discussão de resultados

Como todos os quatro sistemas de recomendação foram modelados como problemas de regressão, foram calculados para cada algoritmo o erro absoluto médio (MAE), o erro absoluto médio normalizado (NMAE), a raiz do erro quadrático médio (RMSE) e a raiz do erro quadrático normalizada (NRMSE), todos discutidos na seção 1.5.1 do Capítulo 1. Uma vez que essas métricas calculam o erro médio entre o valor real da avaliação e a estimativa apresentada pelo modelo, todas precisam ser minimizadas. O MAE apresenta o erro na mesma escala do conjunto original de avaliações, ou seja, de 1 a 5. O NMAE é normalizado, portanto vai de 0 a 1 e quanto mais próximo de zero, melhor. O RMSE também possui a mesma escala do conjunto original de avaliações, porém penaliza de maneira mais acentuada as maiores diferenças entre as previsões e os valores reais. O NRMSE é o RMSE normalizado, portanto também vai de 0 a 1 e deve ser mais próximo de zero. Os resultados são apresentados na Tabela 16.

Algoritmo	**MAE**	**NMAE**	**RMSE**	**NRMSE**
KNN filtro colaborativo	0,7645	0,1911	1,0087	0,2522
SVD filtro colaborativo	0,7686	0,1921	0,9456	0,2364
KNN baseado em conteúdo	0,7859	0,1965	0,9536	0,2384
RNA baseada em conteúdo	0,7962	0,1991	0,9503	0,2376

Tabela 16 – Comparação de métricas de precisão entre modelos

Em todos os modelos utilizamos o método da validação cruzada *5-fold*, ou seja, o conjunto original de dados utilizado para treinar os algoritmos foi dividido em 5 subconjuntos mutuamente exclusivos (sem nenhuma sobreposição entre si), e o algoritmo foi treinado usando cada um desses subconjuntos de dados de treinamento e testado nos 5 subconjuntos de dados de teste correspondentes. As métricas de precisão foram então

calculadas para cada subconjunto, e a média entre os cinco valores foi considerada como a média final do indicador no conjunto de dados para aquele algoritmo.

Podemos concluir a partir das métricas na Tabela 16 que, enquanto o sistema de filtro colaborativo construído com o algoritmo dos k-vizinhos mais próximos (KNN) possui um erro médio total menor do que os outros, sua métrica NRMSE é a maior entre todos, indicando que as previsões geradas são pouco uniformes, com pontos de grande discrepância entre o valor previsto e o valor real da avaliação. Em contrapartida, o sistema de filtro colaborativo utilizando a decomposição em valores singulares (SVD) tem o segundo menor NMAE e o menor NRMSE, indicando um que o erro médio total é um pouco maior do que o KNN filtro colaborativo, porém o sistema gera previsões de maneira mais uniforme, ou seja, minimiza as grandes discrepâncias entre os valores reais e os valores previstos pelo sistema.

No geral, os sistemas de filtro colaborativo parecem apresentar melhores resultados do que os sistemas baseados em conteúdo. No entanto, nenhum sistema se destacada fortemente, uma vez que as diferenças nas métricas de previsão dos quatro sistemas são pequenas.

Além das métricas de precisão, comparamos também os top 10 itens recomendados pelos quatro sistemas. Um usuário aleatório foi selecionado, e as top 10 recomendações para esse usuário em cada um dos sistemas é apresentada abaixo.

Ranking	**KNN colaborativo**	**SVD colaborativo**	**KNN conteúdo**	**RNA conteúdo**
1	A	A	A	B
2	B	E	F	A
3	C	K	B	F
4	D	L	I	D
5	E	M	H	M
6	F	N	K	C
7	G	B	Q	T
8	H	O	R	L
9	I	P	D	K
10	J	D	S	H

Tabela 17 – Comparação de top 10 itens recomendados entre modelos

Os itens A, B e D foram recomendados para o usuário em todos os algoritmos. Os itens F, H e K foram recomendados por três algoritmos. Os itens C, E, I, L e M foram recomendados por dois algoritmos, o os outros nove itens restantes foram recomendados por apenas um algoritmo.

3.4 Evoluções no estudo do problema

Na implementação dos algoritmos acima discutidos o problema foi definido como um problema de regressão, ou seja, cada avaliação prevista será um número real entre 1 a 5. Outra maneira de trabalhar o problema seria analisá-lo como um problema de classificação, onde cada nota de 1 a 5 seria uma classe, algo como 1 sendo equivalente a "não gostei"e 5 equivalendo a "gostei muito". Em um ambiente real, poderia ser realizado um teste A/B onde um grupo de usuários recebe recomendações geradas por um algoritmo de regressão e outro grupo por um algoritmo de classificação. Pode-se comparar a performance dos dois grupos, ou seja, qual grupo com o maior índice de interação entre os usuários e os itens recomendados.

Além disso, a performance dos algoritmos baseados em conteúdo depende muito das variáveis de entrada utilizadas, e isso é especialmente importante uma vez que cada segmento possui variáveis mais e menos relevantes que levam a tomada de decisão por seus clientes. Se as variáveis consideradas no modelo têm pouca relação com a tomada de decisão do usuário, o modelo invariavelmente apresentará resultados ruins. Nas aplicações de sistemas baseados em conteúdo discutidas nesse capítulo todas as variáveis diziam respeito às medidas do usuário, mas podem ser utilizadas variáveis que dizem respeito por exemplo à idade, localização geográfica, o valor médio gasto pelo usuário na plataforma, entre outros.

Ademais, existem variáveis específicas ao algoritmo escolhido que devem ser sintonizadas a fim de aprimorar os resultados. No algoritmo dos k-vizinhos mais próximos, por exemplo, podemos variar a função utilizada para calcular a similaridade entre usuários ou itens, além de refinar o número k de usuários a serem considerados na vizinhança de um certo usuário. Na construção de uma rede neural, podemos trabalhar funções de ativação diferentes da função sigmóide, como a função *ReLu* a função *Softmax*, usada na camada de saída para problemas de classificação. Além disso, é possível modificar a própria topologia da rede neural, assim como a função utilizada para calcular o erro e ajustar os pesos.

Finalmente, pode-se criar um sistema de recomendação que utilize vários dos algoritmos discutidos em conjunto. É possível, por exemplo, aplicar a decomposição por valores singulares a fim de mitigar uma matriz esparsa, a transformando em uma matriz de menor dimensionalidade completamente especificada, e então aplicar essa nova matriz em outro algoritmo. Isso poderia ser feito antes da aplicação do método dos k-vizinhos mais próximos no sistema de filtro colaborativo, de forma que a determinação da vizinhança de um usuário é realizada com uma matriz completamente especificada ao invés da matriz esparsa original. Outra possibilidade para sistemas de filtragem colaborativa é criar várias matrizes de avaliações de m usuários a n itens, cada uma sendo gerada a partir de um algoritmo diferente, e usar todas as previsões a_{ij} de um usuário i a um item j de cada algoritmo e combiná-las, usando por exemplo uma média ponderada, a fim de encontrar a

previsão final.

Conclusão

Nessa dissertação apresentamos uma revisão sobre os principais modelos de sistemas de recomendação, dando destaque aos conceitos matemáticos que sustentam os algoritmos computacionais utilizados na construção desses modelos. Como forma de ilustrar os modelos com base em dados reais, implementamos os algoritmos e realizamos quatro estudos de casos para comparar os resultados.

O estudo contemplou uma revisão sobre as duas estratégias mais utilizadas na construção de sistemas de recomendação, a filtragem colaborativa e a filtragem baseada em conteúdo. Do ponto de vista computacional, foram enfatizados os principais algoritmos de aprendizado de máquina supervisionado, com especial atenção para os aspectos matemáticos da regressão linear, redução de dimensionalidade e decomposição SVD de matrizes, K-vizinhos mais próximos, árvores de decisão e, finalmente, redes neurais artificiais.

Como forma de aplicar os conceitos estudados, construimos modelos utilizando alguns dos algoritmos e propomos sistemas de recomendação baseados em um caso real, com o objetivo de recomendar produtos para um site de venda de vestuário online. Os resultados de cada algoritmo foram analisados sob a perspectiva dos indicadores de erro absoluto médio (MAE) e a raíz do erro quadrático médio (RMSE).

Um assunto interessante e que pode ser abordado em trabalhos futuros é o relevante problema de recomendação em redes sociais, conceito usado por exemplo para recomendar novas amizades ou conteúdo em plataformas como o Facebook, Instagram, Youtube. Em geral, esses sistemas de recomendação são construídos a partir das relações estruturais entre usuários, classificando os nós e arestas da rede formada e baseando-se em características típicas de redes (ou grafos), como o número de conexões que chegam a um nó específico.

Outra perspectiva muito relevante é o aspecto ético envolvido na implementação de um sistema de recomendação, uma vez que esse tipo de sistema é capaz de incentivar e reforçar certos padrões de comportamento. Poderia um sistema de recomendação controlar as ações do usuário? Se sim, quais os perigos inerentes ao uso de sistemas de recomendação? Existe um problema de invasão de privacidade no momento da coleta dos dados que alimentam os sistemas de recomendação? Todos esses questionamentos são de suma importância para indivíduos ou empresas que almejam utilizar sistemas de recomendação e podem suscitar trabalhos acadêmicos relevantes, por exemplo, na área de *fairness*.

Do ponto de vista pessoal, a execução desse trabalho trouxe grande satisfação, uma vez que nos permitiu um estudo mais aprofundado sobre os conceitos matemáticos que embasam alguns dos algoritmos de inteligência artificial mais utilizados hoje em dia.

Nas empresas por onde passei foram algumas as vezes em que utilizamos algoritmos de aprendizado de máquina a fim de resolver um problema real, mas admito que muitas dessas aplicações eram realizadas sem um entendimento adequado dos conceitos matemáticos que regram o comportamento desses algoritmos. Além disso, vi grande valor em direcionar o estudo a problemas de recomendação, assunto bastante discutido tanto em círculos acadêmicos quanto corporativos.

Referências

ACADEMY, C. *Training Set vs Validation Set vs Test Set*. 2021. Disponível em: <https://www.codecademy.com/articles/training-set-vs-validation-set-vs-test-set>. Citado na página 35.

AGGARWAL, C. C. *Recommender Systems: The textbook*. [S.l.]: Springer, 2016. Citado 21 vezes nas páginas 17, 19, 20, 21, 24, 25, 26, 27, 29, 31, 36, 43, 45, 46, 51, 52, 53, 59, 78, 85 e 100.

BATISTA, G. E. A. P. A.; SILVA, D. F. How k-Nearest Neighbor Parameters Affect its Performance. In: *Argentine Symposium on Artificial Intelligence*. [S.l.: s.n.], 2009. p. 1–12. Citado na página 58.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006. Citado na página 32.

BOLDRINI, J. L.; COSTA, S. I. R.; FIGUEIREDO, V. L.; WETZLER, H. G. *Algebra Linear*. [S.l.]: HARBRA Ltda, 1986. Citado na página 47.

DOMKE, J. *Statistical Machine Learning: Trees*. 2010. Disponível em: <https://people.cs.umass.edu/~domke/courses/sml2010/08trees.pdf>. Citado 3 vezes nas páginas 70, 71 e 72.

GEBREMESKEL, G. G.; VRIES, A. P. de. Recommender systems evaluations: Offline, online, time and a/a test. *Working Notes of the Conference and Labs of the Evaluation Forum 2016*, p. 642–656, 2016. Citado na página 26.

GOLDBERG, D.; NICHOLS, D.; OKI, B. M.; TERRY, D. Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, v. 35, n. 12, p. 61–70, 1992. Citado na página 18.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: The MIT Press, 2016. Citado 4 vezes nas páginas 17, 73, 74 e 78.

GUNDERSEN, G. *Proof of the Singular Value Decomposition*. 2018. Disponível em: <https://gregorygundersen.com/blog/2018/12/20/svd-proof/#1-gram-matrices-as-positive-semi-definite>. Citado 2 vezes nas páginas 47 e 49.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. [S.l.]: Springer, 2017. Citado 3 vezes nas páginas 17, 70 e 73.

HUG, N. *k-NN inspired algorithms*. 2015. Disponível em: <https://surprise.readthedocs.io/en/stable/knn_inspired.html>. Citado na página 107.

HUI, J. *Machine Learning — Singular Value Decomposition (SVD) Principal Component Analysis (PCA)*. 2019. Disponível em: <https://bit.ly/3rvUXZk>. Citado 2 vezes nas páginas 49 e 50.

KANE, F. *Building Recommending Systems with Machine Learning and AI*. [S.l.]: Sundog Education, 2018. Citado 8 vezes nas páginas 16, 17, 26, 27, 28, 29, 30 e 58.

KONSTAN, J. A.; RIEDL, J. Recommender systems: from algorithms to user experience. *Springer Science+Business Media B.V. 2012*, 2012. Citado na página 18.

KOREN, Y.; BELL, R.; VOLINSKY, C. Matrix Factorization Techniques for Recommender Systems. *Computer*, v. 42, n. 8, p. 42–49, 2009. Citado 3 vezes nas páginas 22, 24 e 108.

KUBAT, M. *An Introduction to Machine Learning*. [S.l.]: Springer, 2017. Citado na página 32.

LAKSHMANAN, S. *How, When and Why Should You Normalize / Standardize / Rescale Your Data?* 2019. Disponível em: <https://medium.com/@swethalakshmanan14/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff>. Citado na página 42.

LAST, M.; MAIMOM, O.; MINKOV, E. Improving stability of decision trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 2002. Citado na página 69.

LI, S. *Evaluating A Real-Life Recommender System, Error-Based and Ranking-Based*. 2019. Disponível em: <https://towardsdatascience.com/evaluating-a-real-life-recommender-system-error-based-and-ranking-based-84708e3285b>. Citado na página 29.

MACKENZIE, I.; MEYER, C.; NOBLE, S. *How retailers can keep up with consumers*. 2013. Disponível em: <https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers>. Citado na página 20.

MARGALIT, D. *Interactive Linear Algebra*. [S.l.]: Georgia Institute of Technology, 2019. Citado na página 39.

MARSLAND, S. *Machine Learning: An Algorithmic Perspective*. [S.l.]: CRC Press, 2009. Citado 5 vezes nas páginas 17, 32, 69, 70 e 73.

MISRA, R.; WAN, M.; MCAULEY, J. Decomposing fit semantics for product size recommendation in metric spaces. *RecSys '18: Proceedings of the 12th ACM Conference on Recommender Systems*, 2018. Citado na página 104.

MOCANU, D. C.; MOCANU, E.; STONE, P.; NGUYEN, P. H.; GIBESCU, M.; LIOTTA, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications 9*, 2018. Citado na página 88.

OLIVEIRA, J. V. de. Estudo da decomposição em valores singulares e análise dos componentes principais. In: . [S.l.: s.n.], 2016. Citado na página 51.

PANDORA. *About the Music Genome Project*. 2019. Disponível em: <https://www.pandora.com/about/mgp>. Citado na página 16.

RESNICK, P.; IACOVOU, N.; SUCHAK, M.; BERGSTROM, P.; RIEDL, J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, v. 1, p. 175–186, 1994. Citado na página 18.

ROKACH, L.; MAIMON, O. *Data mining with decision trees: theory and applications*. [S.l.]: World Scientific, 2015. Citado na página 70.

SCRAPEHERO. *How Many Products Does Amazon Sell? – April 2019*. 2019. Disponível em: <https://www.scrapehero.com/number-of-products-on-amazon-april-2019/>. Citado na página 16.

SHARDANAND, U.; MAES, P. Social Information Filtering: Algorithms for Automating "Word of Mouth". *CHI '95 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, v. 1, p. 210–217, 1995. Citado na página 18.

SONI, D. *Supervised vs. Unsupervised Learning*. 2018. Disponível em: <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>. Citado 2 vezes nas páginas 34 e 35.

STONE, J. V. *Artificial Intelligence Engines: A Tutorial Introduction of the Mathematics of Deep Learning*. [S.l.]: Sebtel Press, 2020. Citado 4 vezes nas páginas 17, 86, 91 e 93.

TONDJI, L. N. *Web Recommender System for Job Seeking and Recruiting*. Tese (Doutorado), 02 2018. Citado na página 22.

WOLFRAM, S. *Data Science of the Facebook World*. 2013. Disponível em: <https://writings.stephenwolfram.com/2013/04/data-science-of-the-facebook-world/>. Citado na página 21.