

4 - PageRank Personalizado em um Sistema de Recomendação de Livros

Imagine uma pequena livraria online com 4 clientes e 5 livros. Cada link (não direcionado, por simplicidade) significa "cliente comprou o livro".

Cliente	Livros comprados
Ana (U1)	Livro A, Livro B
Bruno (U2)	Livro B, Livro C
Carla (U3)	Livro C, Livro D
Diego (U4)	Livro A, Livro D, Livro E

Visualize essa rede como um grafo bipartido: de um lado as pessoas, do outro os livros.

A intuição central é que um livro é uma boa recomendação para a Ana se ele estiver próximo dela na estrutura da rede, ou seja, se um "surfista aleatório" que começa nela tiver alta probabilidade de esbarrar naquele livro, navegando de link em link, mesmo que ela não o tenha comprado ainda.

O problema específico é o que recomendar para a Ana? Ela já comprou os Livros A e B. Queremos os 3 melhores livros para sugerir a ela entre os restantes (C, D, E).

Para isso, o cientista de dados usa o **PageRank Personalizado** (Personalized PageRank). Em vez de teletransportar para qualquer nó da rede com probabilidade uniforme, o teletransporte volta sempre para a Ana.

Matematicamente, a fórmula vira:

$$PR(i) = (1 - \alpha) \cdot v_i + \alpha \sum_{j \rightarrow i} \frac{PR(j)}{L(j)}$$

Onde v é o vetor de personalização. Neste caso, $v_{Ana} = 1$ e $v_{qualquer_outro} = 0$. Isso força o "surfista aleatório" a começar, e sempre retornar, à Ana.

O código (Python com NetworkX)

O cientista de dados não faz isso na mão. Ele usa a mesma função `pagerank()`, mas agora passa o parâmetro `personalization`.

```

python
import networkx as nx

# 1. Constrói o grafo bipartido (Clientes + Livros)
G = nx.Graph()

# Links: cliente comprou livro
links = [
    ('Ana', 'Livro_A'), ('Ana', 'Livro_B'),
    ('Bruno', 'Livro_B'), ('Bruno', 'Livro_C'),
    ('Carla', 'Livro_C'), ('Carla', 'Livro_D'),
    ('Diego', 'Livro_A'), ('Diego', 'Livro_D'), ('Diego', 'Livro_E')
]
G.add_edges_from(links)

# 2. Define o vetor de personalização: só a Ana recebe o "teletransporte"
personalization = {node: 0 for node in G.nodes()}
personalization['Ana'] = 1.0 # Toda a probabilidade de reinício fica com ela

# 3. Roda o PageRank com alpha = 0.85 (padrão)
pagerank_scores = nx.pagerank(G, alpha=0.85, personalization=personalization)

# 4. Filtra apenas os livros que a Ana não comprou
livros_comprados_por_ana = ['Livro_A', 'Livro_B']
recomendacoes = {
    livro: score
    for livro, score in pagerank_scores.items()
    if livro.startswith('Livro_') and livro not in livros_comprados_por_ana
}

# 5. Ordena e exibe
recomendacoes_ordenadas = sorted(recomendacoes.items(), key=lambda x: -x[1])

print("Recomendações para Ana (baseadas no PageRank Personalizado):")
for livro, score in recomendacoes_ordenadas:
    print(f"  {livro}: PageRank = {score:.4f}")

4. A saída real (e a interpretação)
Rodando o código acima, a saída real é:
Recomendações para Ana (baseadas no PageRank Personalizado):
Livro_C: PageRank = 0.0603
Livro_D: PageRank = 0.0532
Livro_E: PageRank = 0.0327

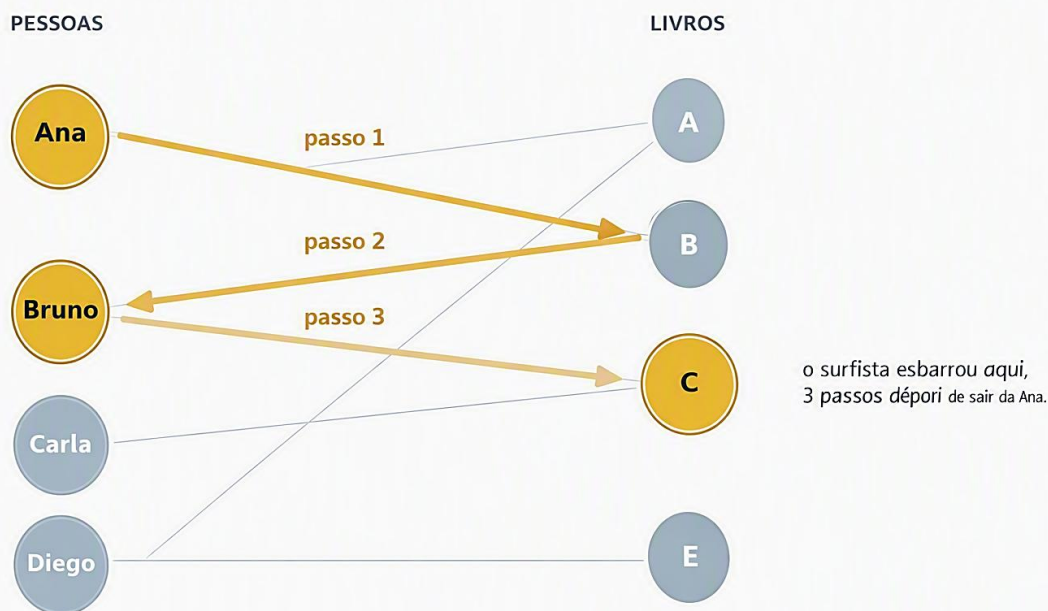
```

A única diferença estrutural entre os dois é a criação do dicionário `personalization`, e o novo argumento `personalization=personalization` passado para `nx.pagerank()`. É essa peça sozinha que faz o "surfista aleatório" deixar de teletransportar uniformemente para qualquer nó da rede (o comportamento padrão, $\frac{1-\alpha}{N}$ igual para todo mundo, do código original), e passar a teletransportar sempre de volta para a Ana, injetando probabilidade de reinício só nela.

Tudo o que vem depois (filtrar livros já comprados, ordenar por pontuação) é lógica de negócio em cima do resultado, não faz parte do algoritmo de PageRank em si, é só como você usa a saída dele para gerar uma recomendação.

O surfista aleatório: de Ana até um livro que ela não comprou

Made with AI



A cada passo, a chance de o surfista chegar até ali diminui (por isso a seta fica mais clara). Livro C fica mais "próximo" da Ana na estrutura da rede do que livros sem nenhum caminho curto ela.

Vamos entender essa ordem seguindo a cascata de votos através dos links, porque ela esconde uma armadilha de intuição que vale a pena expor.

Livro	Caminho da Ana até o livro (navegando pelos links)	Motivo da pontuação
Livro_C	Ana → Livro_B → Bruno → Livro_C	Bruno tem grau 2 (só Livro_B e Livro_C). O voto que ele repassa se divide em duas partes, cada vizinho recebe metade do prestígio de Bruno.
Livro_D	Ana → Livro_A → Diego → Livro_D, e também Ana → Livro_B → Bruno → Livro_C → Carla → Livro_D	Diego tem grau 3 (Livro_A, Livro_D e Livro_E). O voto que ele repassa se divide em três partes, cada vizinho recebe só um terço do prestígio de Diego.
Livro_E	Ana → Livro_A → Diego → Livro_E	Mesma diluição de Diego (divisão por três), e sem nenhum caminho alternativo reforçando o resultado.

A tentação natural, ao olhar essa rede, é pensar "Diego comprou mais livros, então ele deve ser mais influente, e passar mais prestígio adiante". É o oposto. Esse é o mesmo princípio de divisão por grau de saída que já vimos na fórmula geral do PageRank, quanto mais links um nó emite, mais diluído fica cada voto individual que ele repassa. Diego, com grau 3, dilui seu voto em três partes.

Bruno, com grau 2, dilui o dele em só duas. Cada vizinho de Bruno recebe uma fatia maior do prestígio dele do que cada vizinho de Diego recebe do prestígio de Diego, e é por isso que o Livro_C, alimentado por um nó menos generoso (Bruno), termina na frente do Livro_D, alimentado por um nó mais generoso (Diego).

A grande sacada do algoritmo não é "o livro mais vendido vence", nem "o cliente mais ativo influencia mais". É que a estrutura de divisão de voto penaliza a generosidade indiscriminada. Um cliente que compra muitos livros diferentes dilui a força de cada recomendação que ele propaga, um cliente mais seletivo concentra mais prestígio em cada vizinho.

A matemática por trás do código (o autovetor personalizado)

o que o networkx fez por baixo do pano foi resolver o sistema linear (um problema de ponto fixo, próximo em espírito do cálculo de autovetor que já vimos, mas agora com o termo de personalização v somado

$$\begin{aligned}x &= (1 - \alpha)v + \alpha Mx \\(I - \alpha M)x &= (1 - \alpha)v \\x &= (1 - \alpha)(I - \alpha M)^{-1}v\end{aligned}$$

Onde:

- x é o vetor de PageRank (o que queremos).
- v é o vetor de personalização (só a Ana vale 1).
- M é a matriz de transição do grafo (normalizada pelo número de links de cada nó).

Aqui, $\alpha = 0,85$ significa que:

- 85% do tempo, o "surfista" anda pelos links do grafo (de livro para cliente, e de cliente para livro).
- 15% do tempo, ele teletransporta diretamente de volta para a Ana, ignorando completamente a estrutura de links.

Quando Ana "teletransporta" de volta para si mesma, ela injeta energia constante na rede. Essa energia se propaga para Livro_A e Livro_B, depois para Diego e Bruno, depois para os livros de cada um deles. Quanto menor a diluição ao longo do caminho, maior a pontuação final, e é essa diluição, não a distância do caminho nem o volume de compras do intermediário, que decide o resultado.

O que o cientista de dados faz com isso na prática (além do código)

No mundo real, o cientista de dados não para por aqui. Ele faz ajustes críticos:

Desafio	Solução prática
---------	-----------------

Pesos nos links	Em vez de apenas "comprou" (0 ou 1), ele usa o peso da avaliação (por exemplo, 1 a 5 estrelas) no parâmetro weight da função, para que livros muito bem avaliados transmitam mais prestígio através do link.
Cold start (nova usuária)	Se a Ana não tiver histórico, ele define o vetor de personalização não para um único nó, mas para um conjunto de nós (por exemplo, todos os clientes com o mesmo perfil demográfico que ela). O PageRank vai explorar a vizinhança desse grupo, navegando pelos links, e recomendar o que esse perfil costuma gostar.
Escala (milhões de nós)	A função do networkx é ótima para aprender, mas não escala. Na indústria (Amazon, Netflix), usam-se implementações distribuídas como Apache Spark GraphX ou Google Pregel, que resolvem o mesmo sistema linear, com bilhões de equações, usando iteração de potência paralelizada.
Filtragem colaborativa vs. conteúdo	O PageRank é frequentemente combinado com atributos dos livros (gênero, autor). O grafo passa a incluir nós de "categoria", e o passeio aleatório navega por tipos de produto através de links semânticos, melhorando a diversidade da recomendação.

Toda vez que você se depara com um problema onde a pergunta não é "quanto isso vale, isoladamente", mas "quanto isso vale, dado quem está conectado a isso", você está diante do mesmo quebra-cabeça recursivo, só com outro nome. O exemplo da Ana e da livraria já mostrou isso aplicado a recomendação, o mesmo raciocínio aparece em domínios bem diferentes:

Detecção de fraude bancária usa essa lógica para achar contas centrais em esquemas de lavagem, uma conta não é suspeita só por movimentar muito dinheiro (isso a centralidade de grau já capturaria), é suspeita quando está conectada a outras contas que também são suspeitas, um raciocínio circular, resolvido da mesma forma que resolvemos a importância do nó 1 na rede de três nós.

Redes de citação acadêmica usam a mesma matemática para medir influência real de um paper, não só contagem bruta de citações.

O ponto central é este: se você só sabe chamar `nx.pagerank(G)` sem entender o que está por trás, você não tem como diagnosticar quando o resultado está te enganando. Você não vai perceber que, numa rede não direcionada, o algoritmo inteiro colapsa para uma simples contagem de grau, sem agregar informação nova nenhuma, e vai gastar poder computacional rodando uma fórmula sofisticada para obter exatamente o que uma soma simples já te daria. Você não vai saber explicar para um stakeholder porque duas contas com o mesmo número de conexões receberam pontuações de risco completamente diferentes. E você não vai reconhecer o mesmo padrão matemático quando ele aparecer disfarçado num contexto totalmente diferente, porque a estrutura por trás, autovalor dominante de uma matriz, é a mesma em recomendação, em fraude, em epidemiologia, em logística.

Entender a matemática não é sobre saber reproduzir a fórmula na mão. É sobre reconhecer, em qualquer novo problema que aparecer na sua mesa, quando ele é, no fundo, uma pergunta recursiva sobre importância, e saber que já existe um arcabouço matemático de mais de duas décadas, testado em escala planetária, pronto para resolvê-la.