02

Gerando recomendações baseado em um usuário

Transcrição

Agora que já somos capazes de calcular a distância entre um usuários e os outros do sistema, a execução do nosso código começa a ficar mais lenta. Portanto,vamos elaborar uma maneira de testarmos o nosso sistema, que intitularemos como "Parâmetros para teste".

Primeiramente, vamos refatorar a função mais_proximos_de(voce_id), adicionando um parâmetro n que definiremos, por padrão, como None. Esse parâmetro será passado para frente na função distancia_de_todos(voce_id, n = n).

A função distancia_de_todos(voce_id) também vai receber o n, que por padrão será vazio (None). Nela, após armazenarmos todos_os_usuarios, faremos uma condicional que verificará se um n foi passado. Nesse caso, todos_os_usuarios será sobrescrito com um filtro que itera somente até o enésimo elemento (todos_os_usuarios[:n]).

Agora, quando executarmos a função mais_proximos_de(), só serão analisados os elementos até. Se não passarmos nenhum parâmetro para n, todos os usuários serão analisados.

Para testarmos, faremos mais_proximos_de(1, n = 50) para iterarmos somente nos 50 primeiros elementos. Após a execução, teremos apenas dois usuários mais próximos de usuario1, e o resto deles não terá filmes em comum, recebendo o valor 100.000 que definimos na aula passada. Repare, então, que ainda que tenhamos feito essa definição, esses usuários ainda vão aparecer nas nossas listas em alguma situação.

Nesse momento, após termos conseguido chegar a algum resultado e entender a ordenação, vamos eliminar esses usuários indesejados. Se quiséssemos fazer isso, por exemplo, não adicionando as distâncias iguais a 100.000 na nossa lista. Porém, esse seria um código mais complexo.

Outra abordagem mais simples é, na condicional que criamos em distancia_de_usuarios(), ao invés de retornarmos [usuario_id1, usuario_id2, 100000], retornarmos nada (None).

Agora, em distancia_de_todos(), utilizaremos a função filter() do Python, que recebe dois parâmetros: um elemento de comparação e o objeto a ser filtrado. Portanto, faremos filter(None, distancias) para removermos todos os None da nossa variável. Para efetivamente aplicarmos o filtro, podemos criar uma lista baseada no resultado do filtro, o que é feito com list(filter(None, distancias)). Armazenaremos esse retorno na variável distancias.

Executando novamente mais_proximos_de(1, n = 50), teremos somente dois resultados. Além disso, como não estamos passando por todos os usuários do sistema, a execução do código tornou-se bem mais rápida. Porém, lembre-se que n = 50 não quer dizer que queremos os 50 usuários mais próximos, mas sim que estamos buscando os usuários mais próximos dentro dos 50 primeiros do nosso dataset.

Portanto, n é uma letra que pode nos passar a impressão errada, e não é o ideal. Afinal, abreviações em nomes de variáveis sempre são complicadas. Como queremos analisar somente um subconjunto dos usuários, vamos modificá-lo para o parâmetro numero_de_usuarios_a_analisar, e modificaremos as outras ocorrências de n para esse novo parâmetro.

Agora queremos encontrar o usuário mais próximo de usuario1, e utilizá-lo para fazer as recomendações. Começaremos criando uma variável voce = 1, representando o usuario1. Em seguida, faremos mais_proximos_de(voce, numero_de_usuarios_a_analisar = 50), atribuindo o retorno a uma variável similares. Pegaremos, então, similares.iloc[0] para efetuarmos um acesso aleatório à linha 0 do conjunto. Isso nos retornará o valor voce (que é o usuario1), distancia (3.04) e o

nome do que é retornado, que é o índice. Como queremos esse valor, faremos similares.iloc[0].name, retornando o valor 4, que é o nosso usuário mais similar.

Em seguida, faremos notas_do_similar = notas_do_usuario(similar), armazenando as notas do usuário mais próximo de usuario1. Em seguida, teremos que remover as notas atribuídas a filmes que usuario1 já assistiu. Para isso, criaremos uma nova variável notas_de_voce = notas_do_usuario(voce), e sobrescreveremos notas_do_similar com notas_do_similar.drop(notas_de_voce).

Porém, lembre-se que notas_de_voce e notas_do_similar são duas colunas, filmeId e nota. Como a função drop() procura sempre no índice, precisaremos passar os ids dos filmes, e não o dataframe inteiro. Por isso, faremos notas_do_similar.drop(notas_de_voce['filmeId']), certo?

Na verdade temos que tomar um outro cuidado aqui. No Pandas, o índice não é considerado uma coluna, e portanto não poderá ser acessado dessa forma. Para executarmos nosso código da maneira correta, criaremos uma variáfel filmes_que_voce_ja_viu, recebendo notas_de_voce.index. Em seguida, faremos notas_do_similar.drop(filmes_que_voce_ja_viu).

Executando nosso novo código... teremos outro erro. Isso porque alguns dos filmes listados em usuario1 não estã contidos no outro usuário. Como já discutimos anteriormente, o drop() retorna um erro quando algum valor a ser removido não existe no dataframe. Para resolvermos esse problema, adicionaremos o parâmetro errors=Ingnore.

Agora, finalmente conseguiremos a lista de filmes que podem ser recomendados. Faremos um notas_do_similar.sort_values("nota", ascending=False) para obtermos essa lista da melhor nota para a pior, e a atribuiremos para uma variável recomendacoes.

Colocaremos todo esse processo em uma função sugere_para(voce), que pega as notas de um usuário (voce) e extrai os filmes que ele já assistiu da lista do usuário mais similar. Então, ela ordena os filmes que a pessoa similar já assistiu a partir da maior nota, e retorna as recomendacoes.

Repare que essas recomendações estão com o filmeId e a nota. Para facilitar nossa análise, vamos colocar os nomes dos filmes. Para isso, faremos recomendacoes.join(filmes). Também moveremos o parâmetro numero_de_usuarios_a_analisar para a definição dessa função, configurando que seu valor padrão é None.

Agora, se rodarmos sugere_para(1, numero_de_usuarios_a_analisar=50).head(), lembrando que estamos buscando pessoas similares apenas nos primeiros 50 usuários do dataset, teremos uma lista de 5 filmes que podem ser recomendados para o usuario1, como Willow, Old Boy, Clear and Present Danger, Gladiator e Shrek 2.

Feito esse teste, vamos executar com todos os usuários do dataframe, fazendo simplesmente sugere_para(1).head(). Esse processo demorará mais tempo, mas dessa vez encontraremos o usuário mais próximo de usuario1 em todo o dataset, utilizando a função de distância que criamos, e teremos as 5 melhores recomendações.