03

Ordenando usuários por distância e lidando com casos extremos

Transcrição

Fomos capazes de calcular a distância entre um usuários e todos os outros do sistema, mas apareceram alguns resultados estranhos. Por exemplo, a distância entre o usuário e ele mesmo é 0, o que faz sentido, afinal a distância entre um ponto e ele mesmo em uma métrica é sempre 0. Porém, a distância entre o usuario1 e o usuario2 também foi 0... assim como a distância entre usuario1 e usuario3 ou usuario5.

Será que esses usuários realmente deram exatamente as mesmas notas nos filmes que compartilham entre si? Vamos analisar. Faremos `notas_do_usuario(1).join(notas_do_usuario(5), lsuffix="_1", russifx="_5")` para criarmos uma tabela com os índices dos filmes e as notas do usuario1 e do usuario5.

Veremos, então, que o usuario5 não assistiu a nenhum dos filmes que usuario1 assistiu. Portanto, se fizermos um dropna(), removeremos todas as linhas. Se não sobra nenhuma informação, a distância entre esses dois pontos é, por definição, 0.

Isso provavelmente também acontece com muitos outros usuários cuja distância para usuario1 é 0. Poderia ser que algum desses realmente tivesse as exatas mesmas notas que usuario1, mas isso não é o caso pelo menos para usuario2 e usuario3.

Gostaríamos, então, de estabelecer uma quantidade mínima de filmes em comum para quando calcularmos a distância entre os usuários. Afinal, se dois usuários só assistiram a filmes diferentes, não parece ter nada em comum em relação a eles... pelo menos para o nosso sistema.

Agora que entendemos o problema, queremos que os usuários sem filmes em comum sejam colocados bem distantes um do outro. Isso poderia ser feito de várias maneiras, por exemplo eliminando esses resultados do nosso dataframe. Porém, vamos trabalhar de outra forma, ainda mais simples de implementar.

Na nossa função distancia_de_usuarios(), faremos uma condicional que verifica se o tamanho das diferenças é menor que um valor mínimo - por exemplo, 4 filmes em comum. Nesses casos, não iremos considerar esses usuários como próximos, retornando uma distância bem grande, como 100.000. Portanto, faremos return [usuario_id1, usuario_id2, 1000000].

Colocaremos o minimo como parâmetro da nossa função, e o definiremos atualmente como 5. Se você quiser, no futuro, poderá testar outros valores. Executando nosso código, teremos que a distância entre usuario1 e ele mesmo continua 0, mas nos outros casos nos quais os usuários não têm filmes comum, essa distância é muito grande.

Agora, podemos definir uma função mais_proximos_de(voce_id). Nela, primeiro calcularemos a distancia_de_todos(voce_id), que será armazenada em uma variável distancias. Então, faremos distancias.sort_values("distancia") para ordenar esses valores de forma crescente (lembrando que não precisamos definir ascending=True, já que esse já o padrão dessa operação). Sobrescreveremos a variável distancias com os valores ordenados e retornaremos distancias.

Com mais_proximos_de(1), vamos procurar quais são os usuários mais próximos de usuario(1). Como resultado, teremos que o usuário mais próximo de usuario1 é ele mesmo. Em seguida, o usuario214, depois o usuario593, depois o usuario590, e assim por diante. Esses são os usuários mais próximos de usuario1 dada a função de distânicas que definimos, lembrando que estabelecemos um parâmetro mínimo de 5 filmes em comum, uma escolha que traz vantagens e desvantagens.

Para refinarmos nosso dataset, sobrescreveremos novamente a variável distancias, dessa vez com distancias.set_index("outra_pessoa").drop(voce_id). Dessa forma, iremos setar o índice como sendo a coluna outra_pessoa, e removeremos voce_id dessa lista de proximidades, eliminando a distância entre o usuário e ele próprio.

Com isso, fomos capazes de ordenar os usuários de acordo com a função de distâncias que escolhemos.