03

Finalizando uma implementação de KNN

Transcrição

Agora que já conseguimos pegar o usuário mais próximo para basearmos nossas sugestões nas avaliações dele, vamos tentar melhorar esse algoritmo.

Imagine que eu, Guilherme, já assisti a vários filmes na minha vida. Entre eles, existem alguns que muita gente não assistiu, como Os Sete Samurais, que é um filme popular do Japão, mas é um filme de nicho. Ou seja, mesmo que uma pessoa goste muito de filmes de zumbi, assim como eu, esse filme não terá nada a ver com ela. Portanto, seria estranho recomendá-lo.

Sendo assim, quando pegamos as avaliações de somente uma pessoa para criarmos um sistema de recomendação, corremos o risco de existirem alguns nichos muito específicos no meio desses dados.

A ideia é, ao invés de pegarmos somente a pessoa mais similar ao nosso usuário, pegarmos várias - 5, 10, 20, 50, não sei. Se só o Guilherme, que é a pessoa mais parecida com você, gostou de Os Sete Samurais, é uma coisa. Mas de 10 pessoas parecidas com você gostaram de Os Sete Samurais, a história muda. Portanto, tentaremos implementar uma versão simples desse sistema.

A nossa função sugere_para() é um sistema de recomendação baseado em um único usuário. Vamos reaproveitar esse código removendo a definição dessa função e a instrução return. Então, escreveremos as variáveis voce = 1 e numero_de_usuarios_a_analisae = 50.

Agora, não queremos somente um único usuário similar, uma determinada quantidade de usuários mais próximos - os n_mais_proximos. Poderíamos implementar isso depois de ordenarmos os mais próximos, porém, nossa função já é nomeada como mais_proximos_de(), então esperamos que ela já receba os n_mais_proximos = 10, por exemplo.

De volta ao corpo dessa função, ao invés dela devolver as distancias de todos os usuários, ela devolverá somente os primeiros. Faremos isso com return distancias.head(n_mais_proximos), um parâmetro que deverá ser recebido como argumento. Por padrão, vamos defini-lo como 10.

Podemos testar essa função fazendo mais_proximos_de(1, numero_de_usuarios_a_analisar=300). Dessa forma, tentaremos encontrar os 10 usuários mais próximos de usuario1 dentre os 300 primeiros usuários no dataset. Entretanto, como retorno, teremos apenas 7 resultados. Isso porque, nesse conjunto de 300 elementos, somente 7 possuem similaridade o suficiente (ou seja, correspondem a pelo menos 5 filmes assistidos).

Já se fizermos mais_proximos_de(1, n_mais_proximos = 2, numero_de_usuarios_a_analisar=300), teremos como retorno apenas 2 resultados, justamente como deveria ser. Um detalhe: nesse código, estamos ordenando os resultados para então listar os primeiros elementos, mas poderíamos utilizar alguma função que já pega os primeiros à medida em que vai "ordenando".

Na função mais_proximos_de, faremos n_mais_proximos = n_mais_proximos, e definiremos esse parâmetro como 10 fora da função. Podemos jogar fora a linha similar = similares.iloc[0].name. Se executarmos esse código, teremos como retorno apenas 2 resultados - afinal, dentro dos 50 nos quais estamos pesquisando (numero_de_usuarios_a_analisar = 50) só temos dois que satisfazem os critérios estabelecidos.

Nosso objetivo agora é pegar as notas de todos esses usuários. Começaremos criando uma variável usuarios_similares recebendo similares.index para obtermos os respectivos índices desses usuários - no caso, 4 e 26.

Como o índice do dataframe notas não é o usuário, não poderemos usar iloc[] para localizá-los, sendo necessário usarmos uma query(). Porém, seria um código bem complexo que envolveria, por exemplo, a transformação do array [4, 26] em array string. É uma opção, mas ficaria cada vez maior.

Portanto, faremos notas.set_index("usuarioId"). Isso fará com que a coluna usuarioId se torne o índice, sem conflitos com o fato de termos diversos Ids/linhas repetidos. Em seguida, faremos loc[usuarios_similares]. Com isso, traremos todas as notas desses dois usuários.

Imagine então que o usuario4 assistiu ao filme1 e deu a nota 5. Já o usuario26 assistiu ao mesmo filme, mas deu a nota 3. Vamos assumir, de maneira bem simplificada, que o nosso usuário Guilherme, que é parecido com esses dois, daria a nota 3,5 - ou seja, a média.

Vamos atribuir esse dataset a uma variável notas_dos_similares, e faremos groupby["filmeId"] para agrupar as notas dos filmes. Em seguida, faremos mean()["nota"] para obtermos as médias dessas notas (extraindo a coluna "momento"). Com isso, teremos uma series, que só possui uma coluna.

Existem várias maneiras de transformar uma series em um dataframe. Uma delas é utilizando um array para fazer a seleção em notas_dos_similares.groupy("filmeId").mean()[["nota"]]. Atribuiremos esse dataframe a uma variável recomendacoes, e faremos recomendacoes.sort_values("nota", ascending=False) para ordenarmos as notas de maneira descrescente.

Em seguida, juntaremos as recomendações com os filmes e mostraremos as cinco primeiras utilizando recomendacoes.join(filmes).head(). Repare que, como resultado, teremos filmes completamente diferentes daqueles que obtivemos quando estávamos trabalhando com um único usuário.

Naquela situação, apareciam alguns filmes que pareciam ser de nicho, já que possuíam poucos votos (10, 51 ou 77, por exemplo). Agora que estamos trabalhando com dois usuários, apareceram filmes mais populares, todos eles com mais de 400 votos. Perceba também que a coluna "nota" é a nota da recomendação, e a coluna "nota_media" é a nota média que o filme obteve no dataset do MovieLens. Inclusive, os filmes que obtivemos dessa vez possuem uma média um pouco maior do que da primeira tentativa.

Vamos testar agora um numero_de_usuarios_a_alisar = 300. Antes disso, vamos definir a função sugere_para(voce, n_mais_proximos = 10, numero_de_usuarios_a_analisar = None). O corpo dela será o código que criamos acima, retornando recomendacoes.join(filmes).

Agora, se chamarmos sugere_para(1, numero_de_usuarios_a_analisar = 300).head() para buscarmos os usuários similares em uma amostra maior do dataset, teremos ainda outra seleção de filmes nas recomendações. Podemos também remover o parâmetro numero_de_usuarios_a_analisar para buscarmos em todo o dataset, lembrando que esse processo levará um pouco mais de tempo.

Esse processo que implementamos, no qual estamos buscando os "10 usuários mais similares", ou os "n vizinhos mais próximos", é uma versão um pouco menos detalhada de um algoritmo chamado k-NN, ou "k nearest neighbors", utilizado para várias situações diferentes.

Por isso, ao invés de chamarmos a função de mais_proximos_de(), poderíamos chamá-la de knn(). Nessa versão, o n_mais_proximos na verdade seria k_mais_proximos. Também podemos definir o sugere_para(), fazendo com que ele chame k_mais_proximos, e, onde ele chama mais_proximos_de(), na verdade será knn().

Para melhorarmos esse algoritmo, a ideia é refinarmos as funções de distância e de agrupamento dos k_usuários (como a média) para encontrarmos o recomendador mais otimizado. Podemos trabalhar, por exemplo, com grupos de usuários, grupos de filmes de acordo com o gênero, números maiores ou menores de usuários, descartando filmes que tenham um número menor de votos, etc.