03

Mínimo de usuários com recomendações

Transcrição

Uma última variação que eu gostaria de mostrar para aprendermos como todas as partes do nosso algoritmo são customizáveis e podem ser otimizadas é que, quando trazemos os filmes baseados nos k_mais_proximos, estamos tirando a média das notas desses filmes entre todos os usuários.

Porém, se somente um dos usuários assistiu a um filme e deu nota 5, a média dele será 5, por mais que ele não seja um filme "de nicho" na nossa classificação. Nessa situação, esses filmes talvez não sejam exatamente aqueles que gostaríamos de recomendar, já que parece preferível indicarmos um filme de nota 4,5 que foi assistido por dez usuários semelhantes do que um de nota 5 que só foi assistido por um.

Detalhes como este, ou a função da distância, como agrupar os elementos e como tirar as médias, geram recomendadores diferentes.

Voltaremos então à nossa função sugere_para(), na qual estamos buscando os usuarios_similares, tirando as médias das notas_dos_similares e ordenando essas médias. Antes dessa ordenação, queremos jogar fora os filmes que têm poucas avaliações entre os 10 usuarios_similares.

Primeiro vamos analisar novamente a lista de recomendações. Ao invés de sugerirmos filmes para o usuario8621, vamos sugerir novamente para o usuario1, do qual já conhecemos um pouco do comportamento das recomendações. Como numero_de_usuarios_a_analisar, colocaremos 500.

Nesse ponto, pode acontecer um erro causado pelo drop(voce_id) que escrevemos no knn(). Como não o parametrizamos com errors='ignore', nosso algoritmo não funcionará caso ele não encontre no dataframe o usuario que é passado em sugere_para().

Como resultado, teremos as colunas filmeId e nota, que representa as notas médias dos 10 usuários mais próximos. Agora, queremos saber quantas vezes cada um dos filmes na lista apareceu entre esses usuários. Portanto, na função sugere_para(), faremos notas_dos_similares.groupby("filmeId").count()['nota'], que atribuiremos a uma variável aparicoes.

Como escolhemos uma única coluna em count()['nota'], nosso retorno será uma série com o id do filme e quantas vezes ele apareceu. Teremos, então, que o filme1 foi visto pelas dez pessoas; o filme2 por 6 pessoas; o filme3 por 4; e daí em diante.

Agora temos que aplicar o nosso filtro. Mas quantas pessoas devem ter visto o filme para considerarmos a média dele válida? Se o índice também é o filmeId, podemos juntar os conjuntos recomendacoes e aparicoes. Vamos transformá-los em dataframes e utilizar recomendacoes.joint(aparicoes) para juntá-los pelo índice.

Teremos a coluna filmeId, a nota média e a quantidade de aparicoes. Teremos que tomar cuidado, pois os dois conjuntos contém uma coluna chamada nota, o que poderá nos trazer um erro. Portanto, faremos `recomendacoes.join(aparicoes, lsuffix="_media_dos_usuarios", rsuffixs="_aparicoes_nos_usuarios")`, sobrescrevendo a variável recomendacoes com este novo dataframe.

Nesse momento, usaremos o valor 5 como mínimo de usuários que assistiram ao filme, um número relativamente alto, afinal ele é metade das pessoas que estamos utilizando nas nossas recomendações.

Antes de sobrescrevermos o dataframe recomendacoes, faremos filtro_minimo = k_mais_proximos / 2. Então, faremos recomendacoes.query("nota_aparicoes_nos_usuarios >= %.2f" % filtro_minimo). Dessa forma, estaremos buscando em

nota_aparicoes_nos_usuarios valores iguais ou maiores que o nosso filtro_minimo, com duas casas decimais (já que são números de ponto flutuante). Com isso, sobrescreveremos novamente a variável recomendacoes.

Depois disso, continuaremos ordenando as recomendacoes pelas notas (que agora se chamam nota_media_dos_usuarios) e retornando a junção desse dataframe com filmes. Vamos rodar novamente o nosso algoritmo. Como devolução, teremos alguns filmes interessantes, todos com notas boas, mas nenhum deles com nota 5, o que acontecia quando um único usuário tinha avaliado o filme.

Executaremos novamente o algoritmo, dessa vez para o usuario8621, criado anteriormente, utilizando k_mais_proximos = 10 e pegando as 10 primeiras recomendações. Além disso, executaremos o algoritmo também utilizando k_mais_proximos = 20.

Tanto na primeira quanto na segunda lista, as recomendações parecem fazer sentido em relação aos filmes que o usuário 8621 assistiu. Um detalhe importante é: quanto mais usuários estamos pegando com nosso parâmetro k_mais_proximos, mais aparecem nas recomendações filmes muito populares. Isso porque, já que definimos nosso filtro mínimo como k_mais_proximos / 2, quanto maior o número de vizinhos, maior será o número de usuários que terão que ter visto o mesmo filme.

Outra coisa que está acontecendo é que filmes que foram assistidos pelo usuario8621 estão sendo recomendados. Isso porque, quando modificamos a função sugere_para, nós deixamos de remover o filmes_que_voce_javiu das recomendacoes. Faremos isso com recomendacoes = recomendacoes.drop(filmes_que_voce_ja_viu, errors='ignore').

O resultado são diversos filmes que se encaixam com o perfil de usuário que nós criamos anteriormente. Repare que eu forcei duas notas baixas para os filmes de Star Wars pois eu queria que outros filmes de Star Wars não aparecessem na lista. Se você atribuir notas altas a esses filmes, vai perceber que outros filmes de Star Wars aparecerão nas recomendações - o que faz todo o sentido.