PA1 - Collaborative Movie Recommendation

Laura Abreu Schulz Vieira - 2017015061

KEYWORDS

item-based, collaborative filtering

ACM Reference Format:
Laura Abreu Schulz Vieira - 2017015061. 2021. PA1 - Collaborative Movie Recommendation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Este trabalho é referente à disciplina de Sistemas de Recomendação (2021/1) e tem a proposta de implementar um sistema de recomendação de filmes colaborativo. Devemos, através do dataset *ratings.csv*, aprender sobre os itens e usuários, para prever a nota dada pelos usuários aos itens contidos no dataset *targets.csv*. Minha ideia para resolver o problema foi trabalhar com a clássica abordagem *item-based*, computando a similaridade entre os filmes para prever a nota dada pelo usuário alvo. Testei alguns parâmetros como método de imputação de valores faltantes e qual melhor K referente a vizinhança do item. Os resultados serão discutidos com mais profundidade abaixo.

2 DECISÕES DO PROJETO

2.1 Item-based

A primeira decisão que tomei na execução desse trabalho foi a escolha da estratégia de recomendação. Primeiramente, minha ideia era fazer um SVD, mas tive algumas dificuldades com a implementação. Portanto, decidi tentar fazer a clássica item-based, que é mais simples e intuitiva, e me surpreendi com seus resultados.

2.2 Cálculo de similaridade

Optei pela similaridade Pearson (que já considera as normalizações) ponderada por todas avaliações dos itens em questão, como a similaridade cosseno, assim já trazendo uma noção do nível de confiança do cálculo da similaridade.

2.3 Normalização

Optei por fazer a normalização *mean-centered*, por ser clássica e eficiente, considerando a média de avaliações de cada item.

2.4 Algoritmo e Estruturas de dados

As estruturas de dados que escolhi para armazenar minhas matrizes de utilidade foram dicts de dicts, representando, na verdade, listas encadeadas. Isso, pois a memória RAM da minha máquina não comportou as matrizes completes, e conclui que não havia necessidade por serem muito esparsas. Criei estruturas de dados para armazenar todos as avaliações de itens da perspectiva do usuário e da perspectiva dos itens. Também armazeno em dicts as médias de cada um e quantidade de avaliações de cada item e usuário. Escolhi os dicts por terem acesso fácil às chaves em praticamente O(1).

Minha memória também não comportou armazear as matrizes de similaridade, portanto as calculo *online*, mas salvo em um dict com uma tupla indicando a dupla de itens comparados as similaridades já calculadas em iterações anteriores, para evitar cálculos desnecessários.

Meu algoritmo principal funciona iterando sobre o DataFrame correspondente ao *targets.csv*. Verifica se não é um caso de cold-start e, se sim, realiza a imputação, se não, itera sobre todas as avaliações feitas pelo usuário alvo e calcula a similaridade entre os itens avaliados por ele e o item alvo. O top-K é formado pelos itens mais similares ao alvo e a predição é calculada realizando uma média ponderada das avaliações dadas pelos usuários aos itens mais similares em relação a similaridade.

2.5 Vizinhança

A vizinhança que escolhi foi K = 20, que representa valor suficiente para colher informação sobre o gosto usuários e evitar ruidos trazidos por itens com similaridade muito baixa em relação ao alvo. Na seção Experimentos e Resultados há uma análise mais aprofundada dessa seleção.

2.6 Imputação

Como há casos de cold-start, tanto de usuários novos quanto de itens novos, e a estratégia clássica item-based não trata por default casos novos, tive que pensar em estratégia de imputação de valores.

Para casos de usuários novos, mas itens já avaliados, imputo com a média ponderada entre a média global (média de todos os itens e todos usuários, que é aprox. 7,29) e a média de avaliação do item em questão. Com a média de avaliação, temos uma noção da opinião popular sobre o filme, entretanto, pode ser que ele seja avaliado por poucas pessoas, então ponderar pela quantidade de avaliações e a média global previne o problema de baixa confiança.

Fiz o equivalente no caso de itens novos, porém com a média de avaliação dada pelo usuário em questão.

E, no caso de ambos serem novos, imputei apenas a média global.

3 ANÁLISE TEÓRICA E EXPERIMENTAL

3.1 Complexidade

A complexidade de criar as listas de adjacências correspondentes à matriz de utilidade é linear no tamanho do dataset de treino. Já a complexidade de calcular as similaridades online é O(nmw) no pior caso, sendo n a quantidade de itens vizinhos e avaliados pelo usuário - que no pior caso é a quantidade de itens - 1, m a quantidade de usuários que avaliaram os itens em comun (dentro do cálculo da similaridade de Pearson) e w o tamanho do dataset de teste.

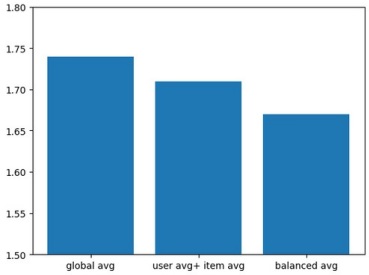
Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3.2 Experimentos e Resultados

- Testei vários métodos de imputação. O melhor no ranking público do Kaggle foi referente à estratégia comentada no tópico imputação. O resultado da imputação fez uma diferença muito grande no meu resultado. O RMSE médio nas três principais situações, mantendo todos os demais parâmetros já previamente setados, ficou conforme a Figura 1 mostra.

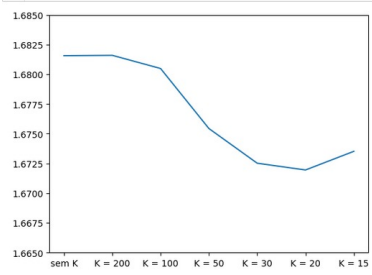
Figure 1: Método de imputação x RMSE



Na figura 1, podemos observar um maior RMSE no primeiro quando tentei o primeiro e mais básico parâmetro default, que é apenas imputar a média global em qualquer caso de cold-start. Temos um resultado de aprox. 1,74. Na segunda tentativa, decidi imputar, no caso de cold-start de usuários a média do item em questão, no caso de itens a média do usuário em questão, e deixar a global apenas para o caso de ambos. Já houve uma melhora significativa na métrica, que caiu para aprox. 1,71. Por fim, a tentativa escolhida foi ponderar pelo número de avaliações com a média global, conforme explicado na sessão anterior, que obteve um melhor resultado de aprox. 1,67. Todos foram testados com a vizinhança K = 20 setada.

- Testei também, depois com a imputação escolhida acima já setada, diversos valores de vizinhança, que também ajudou a melhorar meu resultado, conforme mostra o gráfico abaixo.

Figure 2: K x RMSE



Na figura 2, podemos observar a variação do RMSE de acordo com o K. Testei, respectivamente seguindo a figura, com os Ks: sem K (utilizando todos os itens avaliados pelo usuário), k=200, k=100, k=50, k=30, k=20 e k=15, e obtendo, respectivamente, os RMSEs (computados pelo ranking público do kaggle): 1.68157, 1.68160, 1.68049, 1.67544, 1.67252, 1.67195 e 1.67352. Como podemos observar no gráfico, há pouca diferença entre calcular utilizando todos, 200 ou 100 vizinhos. A maior declinação na curva acontece entre 100-50, e continua com tendência de queda grande entre 50-30. O melhor resultado que obtive foi com k=20, que foi o selecionado por mim. Contudo, já podemos ver na curva que há pouca diferença entre os resultados de 30 e 20. E, por último, testei com k=15, que já é um valor baixo demais e perdemos muita informação importante com ele, de forma que seu resultado já é pior mesmo que com o k=30. A melhora do resultado com a redução da vizinhança é referente à remoção de ruidos.

- Os dois citados acima foram meus principais focos de experimento, pois os outros parâmetros que eu poderia alterar selecionei sem muitos testes, por simplicidade e por acreditar, com base na matéria vista em sala, que possuem desempenho bom o suficiente.
  Meu algoritmo roda em aprox. 150 segundos na minha máquina pessoal, que tem memória RAM 4GB, processador i5 e é um ubuntu 18.04.
  O RMSE final que atingi no ranking público, testado com 48% dos dados finais, é de **1,67195**, com os parâmetros explicitados aqui.

4 DESAFIOS

Houveram muitos desafios implicados em desenvolver este sistema. Um grande desafio para mim foi o tempo. Tive que aprender a manipular bem as estruturas, escolher funções de acesso rápido, evitar trabalhar com DataFrame, etc. Ademais, o cold-start também foi um desafio que me levou a testar diversas formas de imputação de valores nesse caso. Também tive problemas em iniciar. Como dito nas decisões, tentei fazer o SVD, mas não consegui muito bem pensar em como montar a matriz e decidi mudar de abordagem. Já para montar o item based, tive alguns problemas com estruturas e alguns bugs que demorei a descobrir.

5 CONCLUSÕES

O trabalho foi muito engrandecedor para meu aprendizado. Implementar um sistema de recomendação e vê-lo funcionando me ajudou muito a fixar a matéria, ver seus problemas e desafios na prática e a aprender sobre eficiência de estruturas e funções em Python. Ademais, foi um resultado surpreendente. Imaginei que para obter um resultado tal qual o meu precisaria de um algoritmo mais robusto como Matrix Factorization. Entretanto, acredito que meus resultados foram bem satisfatórios para uma abordagem simples como a item-based. Foi bom para ver, empiricamente, que muitas vezes os algoritmos simples, se bem executados, podem trazer resultados bons.

6 REFERÊNCIAS

Slides da matéria.

REFERENCES