Research Challenge #1: Collaborative Product Recommendation

Breno Claudio de Sena Pimenta
brenopimenta@ufmg.br
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil

ABSTRACT

Essa documentação explica as decisões de projeto do trabalho prático desenvolvido para implementar um sistema de predição de notas para usuários e itens utilizando um recomendador de filtragem colaborativa baseado em fatores latentes.

CCS CONCEPTS

• **Information systems → Collaborative filtering**.

KEYWORDS

Sistema de recomendação, Filtragem colaborativa

ACM Reference Format:
Breno Claudio de Sena Pimenta. . Research Challenge #1: Collaborative Product Recommendation. In *DCC049 - Tópicos em Sistemas de Informação: Sistemas de Recomendação, 20 de Dezembro, 2021, Belo Horizonte, MG.* ACM, New York, NY, USA, 2 pages.

1 INTRODUÇÃO

O trabalho prático consiste em implementar um sistema de recomendação colaborativo baseado em memória ou em modelo para realizar a previsão de notas de um conjunto de usuários e itens. Afim de realizar essas previsões, é necessário uma grande quantidade de dados para servir como estimadores de como um usuário poderia se comportar e então prover recomendações relvantes.

2 MODELAGEM COMPUTACIONAL

Inicialmente o problema foi analisado do ponto de vista teórico ao pesquisar o assunto de filtragem colaborativa [1] [6]. Existem diversas formas de realizar a recomendação de itens. As principais formas analisadas para esse trabalho foram: filtragem colaborativa baseada em usuários, filtragem colaborativa baseada em item e modelo baseado em fatores latentes. Como o método de fatores latentes tem melhor resultado de previsão em relação aos demais métodos e o conjuntos de dados oferecido é bem esparso, essa estratégia foi adotada.

Dado o modelo baseado em fatores latentes escolhido, inicialmente foi realizada diversas pesquisas em artigos e livros sobre como esses recomendadores são implementados [4]. Após essa leitura, foi identificado que o primeiro passo seria tratar os dados de entrada para gerar as matrizes esparsas que conteriam os dados a serem previstos.

Esse primeiro passo se mostrou ser extremamente complicado, visto que dado os n usuários e m itens de entrada, gerar uma matriz (n, m) exigia muito da memória, facilmente passando dos 200 Gb. Para tratar esse caso foi possível encontrar algumas soluções [3] [2] [5]. Elas se baseiam no conceito de SVD (Singular Value Decomposition) para fatorar os dados de entrada em em duas outras matrizes cujo produto geraria a matriz original.

A primeira ideia seria utilizar o *numpy.linalg.svd*, porém essa forma se mostrou mais complexa que o desejado. Por isso, foi optado por realizar a mesma decomposição manualmente após realizar um mapeamento dos ids dos usuários e itens aos seus correspondentes índices, definindo-se uma quantidade de fatores para que a matriz não exija muita memória.

Em seguida para realizar o treinamento dos fatores latentes, foi utilizado a técnica do SGD (Stochastic Gradient Descent) para que em cada passada, o algorítimo se encontre mais perto do valor real. A condição de parada desse treinamento são três: o número máximo de passos, o tempo de execução ou a não melhoria do RMSE. O treinamento é feito a partir de um subconjunto dos dados de entrada conhecidos. Assim, o problema foi modelado a partir de uma solução baseada em fatores latentes e SVD e SGD.

3 ALGORITMO IMPLEMENTADO

Para solucionar o problema, após várias pesquisas já citadas, foi possível implementar um algoritmo baseado na solução de Simon Funk [4]. Essa solução se baseia nos seguintes passos:

1. Definição do conjunto de treino e validação: Dado uma entrada R que contém as tuplas (user, item, rating) conhecidas, foram criados dois subconjuntos, ou seja, T que contém os dados de treino, 80% do conjunto de dados conhecidos de R e V que os dados de validação, 20% do conjunto de dados conhecidos de R. T é amostrada aleatoriamente e V são os dados de R que não estão em T.

2. Fatoração de matriz: Dado uma entrada T de tamanho (m, n), essa matriz pode ser decomposta em duas outras matrizes pu e qi de tamanho (m, k) e (n, k) respectivamente.

3. Aprendizado dos fatores: Essa etapa consistem em aprender os fatores latentes para os itens e usuários. Ela consistem em aplicar o seguinte algoritmo com os seguintes parâmetros:

- T: dados de treino (80% do conjunto de dados conhecidos)
- V: dados de validação (20% do conjunto de dados conhecidos)
- $epochs$: passos do SGD
- η: taxa de aprendizado
- k: número de fatores latentes
- λ: fator de regularização
- bu: vetor de bias paras os usuários
- bi: vetor de bias paras os itens
- pu: matriz de fator latente dos usuários
- qi: matriz de fator latente dos itens
- X: média global das avaliações
- $\Delta RMSE$: diferença entre o RMSE atual e o RMSE do passo anterior

- limiar: caso a diferença do RMSE de um novo passo em relação ao anterior seja menor que esse limiar, o recomendador para de realizar o treino visto que está saturado

Algorithm 1: Treinamento dos Fatores Latentes

input : T, V, $epochs$, η, λ, bu, bi, pu, qi, X,$\Delta RMSE$
output: bu, bi, pu, qi

```
init(bu, bi, pu, qi)
foreach epochs do
    foreach u, i, r ∈ T do
        pred ← X + bu[u] + bi[i]
        foreach k do
            pred ← pred + pu[u,k] * qi[i,k]
        err ← r - pred
        bu[u] ← η * (err - λ * bu[u])
        bi[i] ← η * (err - λ * bi[i])
        foreach k do
            pu[u,k] ← pu[u,k] + η *(err * qi[i,k] -
              λ*pu[u,k])
            qi[i,k] ← qi[i,k] + η *(err * pu[u,k] - λ*qi[i,k])
    if Tempo de execução > 4 minutos then
        break
    if ΔRMSE > limiar then
        break
```

4. Predição das notas: Essa etapa consistem utilizar os valores aprendidos do passo anterior e computar as notas para os novos itens. A predição é bem simples, basta para cada usuário u e item i no conjunto targets realizar: $r \leftarrow X + bu[u] + bi[i] + (pu \text{ x } qi)[u, i]$.

4 CÓDIGO IMPLEMENTADO

O código foi feito em python, utilizando-se os pacotes numpy e pandas. Existem dois arquivo de código que compõe a solução do problema:

- **main.py**: que realiza a leitura dos arquivos de entrada e instancia a classe LatentFactorModelRecommender para gerar as recomendações.
- **latentFactorModelRecommender.py**: que implementa a classe LatentFactorModelRecommender cujo objetivo é realizar as operações de aprendizado e predição.

4.1 Arquivo main.py

Nesse arquivo é feito a leitura dos arquivos de entrada a partir da passagem de parâmetros ao executar o programa. O dataset de ratings é dividido em duas partes, 80% dos dados compõe o conjunto que será usado para estimar os fatores latentes e os outros 20% são dados para testar a performance do aprendizado.

A amostragem é feita de maneira aleatória. Esses valores (80 e 20%) foram obtidos após vários testes e leituras das referências já citadas, outros números não proporcionaram a mesma performance e amostragem não aleatória também não se mostrou mais eficaz.

4.2 Arquivo latentFactorModelRecommender.py

Nesse arquivo está implementada a classe que realmente realiza o aprendizado, gera os fatores latentes ótimos e realiza a predição para cada item da entrada. Os detalhes de implementação estão explicados no próprio código. Durante a realização de testes, os valores dos parâmetros de inicialização foram setados para o melhor valor encontrado.

5 PERFORMANCE DO TRABALHO DESENVOLVIDO

O programa foi desenvolvido em linguagem Python e testado em ambiente Linux. O tempo de execução tem uma média de 181s até atingir 13 passadas do SGD e saturar a melhoria do RMSE em uma variação de 0.000001.

A complexidade do algoritmo desenvolvido está relacionado com a complexidade do método _learnFactors que efetivamente computa os valores dos bias e dos fatores latentes utilizando-se da técnica do SGD em e interações. A complexidade de tempo do código para uma entrada R que possui f tuplas (user, item, rating) é O(efk).

6 CONSIDERAÇÕES FINAIS

A partir do trabalho foi possível entender mais sobre sistemas de recomendação e algoritmos para filtragem colaborativa. Com a solução desenvolvida foi possível gerar predições de notas para o conjunto de (usuários, itens) a partir de um outro conjunto conhecido de notas já dadas pelos usuários.

A maior dificuldade do trabalho foi entender como usar corretamente as estruturas de dados para que se obtenha eficiência de processamento e de memória para conseguir executar o sistema em menos de cinco minutos. A partir da solução desse problema, bastou estudar projetos existentes e buscar exemplos na literatura de boas práticas para atingir uma boa performance a um baixo custo de processamento.

Assim, pode-se concluir que foi possível cumprir com os objetivos do trabalho prático proposto.

REFERENCES

[1] Charu C. Aggarwal. 2016. *Recommender Systems: The Textbook* (1 ed.). Springer.
[2] Geoffrey Bolmier. 2021. *Funk-svd.* GitHub repository. https://github.com/gbolmier/funk-svd.
[3] Scott Freitas and Benjamin Clayton. 2017. *Anime Recommender System Exploration: Final Report.* https://github.com/safreita1/Recommender-System.
[4] Simon Funk. 2006. *Netflix Update: Try This at Home.* Project Sifter. https://sifter.org/simon/journal/20061211.html.
[5] Nicolas Hug. 2017. *Surprise, a Python library for recommender systems.* http://surpriselib.com.
[6] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. *Recommender Systems Handbook* (2 ed.). Pearson Education.