Research Challenge #1

[**Luis Antonio Duarte Sousa**]

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

luisduarte.dev@gmail.com

1 Introdução

Este trabalho tem como objetivo desenvolver um sistema de recomendação colaborativo personalizado, utilizando uma abordagem baseada em modelo para a tarefa proposta no Research Challenge 1: Collaborative Product Recommendation. Os sistemas de recomendação são ferramentas essenciais para filtrar e sugerir itens relevantes em ambientes com grandes volumes de dados, como plataformas de comércio eletrônico, serviços de streaming e redes sociais. No contexto desta implementação, o foco é implementar um recomendador colaborativo que utilize técnicas de fatoração de matrizes, a fim de prever as avaliações de produtos por usuários com base em padrões de comportamento coletivo. Neste trabalho, foi utilizada a abordagem de fatoração de matrizes, um método baseado em aprendizado, que decompõe a matriz de avaliações em fatores latentes que representam tanto as preferências dos usuários quanto as características dos itens. O modelo é treinado utilizando gradiente descendente, ajustando as matrizes fatoriais e os vieses para minimizar o erro preditivo.

2 Implementação

2.1 Estruturas de dados

- Dataframe(Pandas): Armazena as avaliações e alvos após a leitura do CSV e pré-processa os dados inicialmente, normalizando e verificando distribuições.
- Matrizes Bi-dimensionais: Armazena os K fatores latentes de cada item/usuário
- Vetores: Armazena os vieses de cada usuário e item.
- Dicionário: Mapeia o ID dos itens e usuários para inteiros ao invés de strings.

2.2 Classes

- **MatrixFactorization**: Classe criada para representar/modularizar todo o processo de recomendação em métodos no qual os principais são:
  - **indexar_ids**: Recebe como parâmetro o dataframe de notas e apartir dele cria dois dicionários que mapeiam o id real em formato string em inteiros sequenciais unicos.
  - **__init__(construtor)**: Recebe como parâmetros o número de fatores latentes (dimensão das matrizes), um alpha, que representa a taxa de aprendizado do modelo (indicando quanto dos novos valores deve ser considerado), um fator de penalização que ajuda a reduzir o overfitting do modelo, o número de iterações do treinamento (episódios) e, por fim, o DataFrame contendo os IDs de usuários, itens e suas respectivas avaliações (Ratings). O método inicializa as variáveis, embaralha os dados para evitar vieses de ordem no treinamento, mapeia os IDs de usuários e itens para inteiros usando o método "indexar_ids"e inicializa as matrizes de fatores latentes e os vetores de vieses para itens (Q) e usuários (P).
  - **ajustar**: O método não recebe parâmetros e ajusta/treina as matrizes de componentes latentes da seguinte maneira:

    O método itera **n_eps** vezes sobre todas as avaliações, ajustando as previsões e minimizando o erro. Para cada avaliação, é calculada uma predição **P**, utilizando a fórmula:

    $$média\_global + (fat\_u \cdot fat\_i) + viés\_u + viés\_i \quad (1)$$

    Após calcular a predição, o erro **E** é determinado pela diferença entre a nota real e a

predição, ou seja:

$$E = nota\_real - P \quad (2)$$

Com base no erro, o modelo ajusta os vieses e os vetores de fatores latentes dos usuários e itens, minimizando o erro de predição. Os vieses são atualizados pelas seguintes equações:

$$[h]\text{aj\_vies\_i} = \text{tx\_aprend} \times (E - penali \times viés\_i) \quad (3)$$

$$\text{aj\_vies\_u} = \text{tx\_aprend} \times (E - penali \times viés\_u) \quad (4)$$

Os fatores latentes dos usuários e dos itens são ajustados conforme as equações:

$$aj\_Q = tx\_aprend \times (E \times fat\_u - penali \times fat\_i) \quad (5)$$

$$aj\_P = tx\_aprend \times (E \times fat\_i - penali \times fat\_u) \quad (6)$$

Esses ajustes são aplicados iterativamente para minimizar o erro entre as previsões e as notas reais, aprimorando a precisão do sistema de recomendação.

– **estimar_para_alvos**: O métodos recebe um dataframe de alvos para predizer as notas e para cada linha calcula a predição com base nas matrizes e vetores pós treino já com erro minimizado.

3 Análise de Complexidade

- Inicialização: Inicializar as matrizes fatoriais P e Q e os vieses tem complexidade **O(n_usuarios×n_fatores+n_itens×n_fatores)**, onde n_usuários e n_itens são os números de usuários e itens, e n_fatores é o número de fatores latentes.
- Treinamento (Gradiente Descendente): Cada passo do gradiente descendente tem complexidade **O(n_fatores)**, dado que envolve o cálculo do produto escalar entre os vetores de fatores latentes. Para nepsneps iterações e navaliacoesnavaliacoes avaliações, a complexidade total do treinamento é **O(n_eps×n_avaliacoes×n_fatores)**.
- Previsão: A previsão para um par (usuário, item) tem complexidade **O(n_fatores)**, pois envolve o cálculo de um produto escalar entre os fatores latentes do usuário e do item.

4 Conclusão

A solução inicial buscou utilizar a abordagem memory-item-based, com normalização por mean-centering para reduzir os vieses dos usuários, combinada com a similaridade do cosseno e o método de agregação weighed average. A implementação foi estruturada como um dicionário de dicionários, visando mitigar os efeitos da esparsidade dos dados. No entanto, essa abordagem se mostrou impraticável devido ao elevado tempo de execução, que se tornou inviável à medida que o volume de dados aumentava.

Diante desse desafio, a solução foi repensada e uma segunda abordagem foi adotada: a técnica FunkSVD. Após pesquisas sobre como lidar de forma eficiente com dados esparsos, optei por essa abordagem, que utiliza a decomposição de valores singulares (SVD) para fatoração de matrizes. Essa técnica se mostrou eficaz tanto em termos de desempenho quanto de qualidade das recomendações.

Os melhores resultados foram obtidos utilizando os seguintes parâmetros: taxa de aprendizado de 0,05, 100 fatores latentes, 20 iterações e 0,02 de regularização. Esse ajuste proporcionou um equilíbrio adequado entre tempo de execução e qualidade das previsões, com um bom controle de overfitting. Além disso, o uso de FunkSVD reduziu significativamente o tempo de execução em comparação com a abordagem baseada em memória, mantendo um consumo de memória eficiente.

5 Referencias

https://jaco-vanderwalt.medium.com/how-to-use-funk-svd-to-recommend-an-offer-a-learning-journey-of-discovery-1b45f7007e52

https://medium.datadriveninvestor.com/how-funk-singular-value-decomposition-algorithm-work-in-recommendation-engines-36f2fbf62cac

https://surprise.readthedocs.io/en/stable/matrix factorization.html

https://towardsdatascience.com/matrix-factorization-in-recommender-systems-3d3a18009881