Practical Assignment 1
Collaborative Filtering Recommendation

Rennan Cordeiro Lima
rennancl@gmail.com
Univesidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais

1 INTRODUÇÃO

Sistemas de recomendações são ferramentas fundamentais para diversos serviços oferecidos no mundo. Eles possibilitam que tais serviços possam oferecer produtos que seja personalizado a cada um de seus usuário e, dessa maneira, melhorar a experiencia deles. Esse aspecto, a personalização, é fundamental para gerar boas recomendações, tendo a filtragem colaborativa um das perspectiva de se personalizar um sistema.

Nesse trabalho, foi desenvolvido um modelo de filtragem colaborativa baseado na fatoração da matriz de relações entre usuários e itens, utilizando para testes um dataset de avaliações de filmes.

2 IMPLEMENTATION

A implementação do trabalho foi feita em C++ com a criação de duas classes, Matrix, que implementa uma matriz com suas devidas operações e uma classe Model, que contém campos, além dos tipos tradicionais, dois ponteiros para o tipo Matrix.

Essas matrizes, representam as matrizes P e Q, as quais são as matrizes geradas após a fatoração da matriz de usuário e item. A primeira, tem o formato $Usurio \times Fator$, em que cada linha é um vetor que representa os pesos de cada usuário para cada fator; a matriz Q, por sua vez, representa a relação desses mesmos fatores para cada item, em que os vetores para item são as colunas.

Cada avaliação de um usuário I para um item J pode ser reconstruída ao multiplicar-se a linha I da matriz P pela coluna J da matriz Q. Essa propriedade é explorada para aprender essas matrizes.

Em essência, os matrizes são inicializadas com um valor fixo, o que gera avaliações distantes das reais. A diferença aritmética entre a avaliação real e calculada é chamada de erro, esse valor é então utilizado para alterar os pesos em cada um desses vetores, aproximando então o produto entre os vetores de fatores e o valor esperado.

Essa atualização é feita a cada época, no fim do treino os erros são mínimos, e o modelo consegue, com sucesso, prever um avaliação não descoberta.

Em relação aos hiper parâmetros, o modelo foi instanciado da seguinte maneira:

- Épocas (N) : 80
- Fatores (K): 30
- Learning Rate (γ): 0.002

Uma observação importante é que, diversos usuários e itens não estão presentes no treino, mas estão presentes no arquivo de "targets", os quais devemos prevê. Basicamente havia três casos, em que usuário não está presente, em que o item não está presente e em que ambos não havia inicialmente. No primeiro caso, a avaliação padrão é a média do item, no segundo a média do usuário e no último a média global dos itens.

Uma otimização implementada é o viés de cada elemento, tanto de itens e usuário. Esse valor foi calculado baseado em quanto os usuário desviam da média global, a intuição é que, caso um usuário tenha avaliações sempre abaixo da média, espera-se avaliações menores que outros com avaliações acima da média, para um mesmo item. Essa implementação melhorou os resultados, como pode ser melhor analisado no decorrer do desse documento.

Além diss

3 ANÁLISE DE COMPLEXIDADE

Temos para a função principal, e que toma mais tempo, é a função de treino na classe Model, pode ser descrita, em relação a complexidade, da seguinte maneira: sendo K o número de fatores do modelo, N o número de épocas e M o tamanho da entrada de treino em linhas, a função de complexidade seria $O(K \times N \times M)$.

4 RESULTADOS

Os resultados foram feitos em diversas etapas do código. Inicialmente, não foram tratados casos em que usuários e itens não estavam descritos no arquivo "ratings.csv", dessa maneira, o modelo não treinava para esses respectivos elementos, além disso, não havia uma implementação para cobrir o viés de cada um deles. Esse primeiro resultado foi submetido ao Kaggle obtendo um RMSE de aproximadamente 1.9. Em seguida, após alterações de hiper parâmetros, em que o número de épocas e de fatores foi aumentado, o erro diminuiu para 1.69.

O resultados diminuíram mais quando foram coberto os casos de usuários e itens inexistentes, e, com a solução descrita na seção de implementação, o erro diminuiu para aproximadamente 1.65. E seguida, o viés foi implementado e o erro abaixou para 1.63, que representa o valor descrito na última submissão no site.

Além desses resultados descrevendo melhorias seguidas de minimização de erro, alguns resultados foram piores, dependendo da configuração de hiper parâmetros. Um número grande de fatores com poucas épocas, por exemplo, teve um erro grande, o que indica $underfitting$.

Por fim, uma última observação é de que testes utilizando um maior número de fatores, ou maior número de épocas demoram mais, pois, como mostrado pela análise de complexidade, esses parâmetros alteram o tempo de execução

Conference'17, July 2017, Washington, DC, USA
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn