Sistema de recomendação para filmes utilizando Content-based

Lucas C. Lima[1]

[1]Universidade Federal de Minas Gerais (UFMG)

lcs.chv@gmail.com[1]

1. Introdução

SsR são algoritmos que estimam ratings (i.e., avaliações positivas ou negativas que os usuários assinalam a itens consumidos) para itens que ainda não foram consumidos pelo o usuário. Estas ferramentas geram recomendações personalizadas a partir de informações passadas, tais como características dos itens consumidos pelo usuário no sistema (e.g., idade do item no sistema, localização, frequência de consumo, dentre outras). Os sistemas de recomendação são classificados de acordo com o método de predição utilizado: baseado em conteúdo, filtragens colaborativas e métodos híbridos. Nesse trabalho estaremos focando apenas no método baseado em conteúdo.O programa a ser implementado receberá um arquivo (.csv) com ratings de usuários a determinados itens de um sistema e um segundo arquivo (.csv) com informações de conteúdo dos itens. Com base nesses dois arquivos deverá ser gerado recomendações a usuários e itens targets.

2. Visão Geral

Para solucionar o problema proposto, foram implementados duas estratégias:

1. ItemBased: Recomendação utilizando o princípio ”Quanto eu gosto de itens similares a I”.
2. Modelo - (Conteúdo): Recomendação utilizando apenas modelo previamente proposto para gerar recomendações.

2.1. Principais decisões tomadas

Segue abaixo algumas das principais decisões de projeto:

- O recomendador foi desenvolvido utilizando a linguagem C++, na versão C++11 2011 Standard.
- Foi utilizado a biblioteca Boost for 64-bit Linux para a realização do parser de cada linha do arquivo de entrada. Disponibilizada em: ”http://www.boost.org/users/history/version_1_62_0.html”.
- Divisão do projeto em três partes: Processamento dos dados, predição e recomendação.
- Implementação da classe LoadInput para realizar leitura e tratamento dos dados de entrada.
- Implementação de vector space model para realização do cálculo de predição utilizando tf-idf e similaridade de cosseno para realização de predição
- Utilização de maps como estruturas de dados.
- Foi utilizado similaridade de cosseno para o cálculo da similaridade entre os itens.

- Implementação do Rocchio.
- Utilização da biblioteca que se encontra no link ”https://github.com/nlohmann/json”para realização do parser do json.
- Utilização do conteúdo dos itens para criação dos vetores de tokens (Gênero, plot).
- Implementação de um recomendador utilizando para realizar a predição um modelo proposto.
- Foi utilizado RMSE na realização dos testes para avaliação das recomendações.
- A saída segue o seguinte formato: UserId:ItemId,Prediction

3. Descrição do recomendador implementado

Nessa seção será apresentado como foram as estratégias adotadas para implementação dos recomendadores baseado em conteúdo. Essas estratégias são:

1. Rocchio recommendation: Cada item é um vector $\vec{i}$ onde cada termo é um componente. Cada user é um vetor $\vec{u}$ onde tem-se cada termo a multiplicação entre os ratings dados por esse user a seus itens.
2. Content Imdb Average: Realização do conteúdo de uma das tags do arquivo de entrada [”imdbRating”] para realizar predições.

3.1. Rocchio

A implementação do algoritmo Rocchio foi realizada da seguinte forma. Primeiramente, utilizamos um parser para identificar os tokens do arquivo content.csv. Para cada um dos itens contidos nesse arquivo, é feito o parser dos conteúdos das [”Gender”, ”Plot]. Em seguida é realizado o tokenizer de cada uma das palavras e armazenado em duas maps, TokensHash e ItemsMap, uma armazenando por tokens-¿itens e itens-¿tokens respectivamente. Feito o armazenamento, é realizado o processamento dos dados, para isso é realizado o cálculo do TF-IDF para cada token em cada documento e armazena-se na item feature matrix. Em seguida, realiza-se o cálculo da user feature matrix. Para a realização desse cálculo, leva-se em consideração apenas os usuários e itens targets, dessa forma pode-se obter uma eficiência maior na hora de realizar as estimações. Para realizar as estimações, foi utilizado a similaridade de cosseno entre o usuário e item alvo usando as duas matrizes user feature matrix e item feature matrix.

3.2. Content Imdb Average

Uma implementação mais simples do que a do algoritmo Rocchio, foi a utilização apenas do conteúdo do Imdb Rating oferecido no conteúdo do json. Dessa maneira, primeiramente realiza-se o parser de cada um dos itens e seus respectivos ImdRatings. No entanto, nem todos os itens possuiam imdb ratings. Uma solução viável foi utilizar a média dos ratings dados pelo usuário. Caso não possua informações de ambos, utiliza-se 6.7 como rating. Esse valor foi refinado através de iterações a partir da média dos ratings de todos os outros usuários. Essa estratégia, apesar de simples, obteve resultados muito superiores a implementação Rocchio.

4. Análise de Complexidade

Na (Tabela 1) apresento os principais símbolos utilizados na análise de complexidade. Será realizado a análise de complexidade de tempo para ambas estratégias implementadas.

Tabela 1. Definição dos símbolos comuns utilizados na análise de complexidade.

Símbolo	Descrição
U	Número de usuários.
I	Número de items
T	Número de ratings a estimar
K	Número de itens no content.csv
N	Número de tokens
M	Número de itens estimados pelo usuário

4.1. Análise do Rocchio recommender

Primeiramente, para realizarmos o parser do arquivo content.csv teremos uma complexidade $O(K)$. Para cada um dos K itens teremos que identificar e passar pro todos os tokens, logo teremos uma complexidade de leitura e identificação de todos os tokens igual a $O(K) * O(N)$. Em seguida, é realizado o cálculo do $Tf * IDF$, para isso passaremos por todos os tokens de cada um dos itens, resultando em complexidade $O(N) * O(K)$. Mais adiante, para realizar o cálculo da user feature matrix será realizado para cada um dos usuários, para cada item que esse usuário deu rating, para cada termo desse item, o cálculo da média dos valores entre user/item e token/item resultando em um limite superior da complexidade do algoritmo de $O(U) * O(M) * O(N)$.

4.2. Análise do recomendador Content ImdbRating Average

A complexidade computacional do algoritmo de recomendação utilizando a estratégia da média imdbrating depende do tempo necessário para realizar a leitura do arquivo content.csv, logo temos complexidade $O(K)$ para passar por cada um dos itens. Em seguida, caso não possua informações sobre o conteúdo do item é gerado a recomendação utilizando a média dos ratings do usuário alvo, o que resulta em uma complexidade $O(U)$. Por fim, temos a complexidade de realizar a recomendação, como o acesso as maps é constante, teremos apenas $O(T)$. A complexidade final desse estratégia é $O(U) + O(K) + O(T)$.

5. Análise de Resultados

5.1. Eficiência

Para realização da análise de eficiência foi feito testes de tempo levado para realizar a recomendação dos itens targets. Essa análise foi efetuada variando o número de itens a serem recomendados. Com isso na figura (Tabela 2) obtivemos os seguintes tempos de execução.

Tabela 2. Imdbrating content average para diferentes números de itens/usuários

Número de Itens alvo	Número de Usuários alvo	Tempo gasto
70000	70000	13.5 s
140000	140000	14,5 s
200000	200000	16,1 s
400000	400000	21,1 s

Como pode ser observado, os tempos de execução pouco se alteram a medida que aumenta-se o número de itens e usuários a estimar. Uma outra forma de avaliação, seria

aumentando o número de itens no content.csv, no entanto, essa avaliação não era possível nesse TP.

5.2. Efetividade

Para realizamos os testes de eficácia era necessário que tivéssemos um gabarito. Para isso, criamos um GroundTruth, que permitisse avaliar quão bom as recomendações que estão geradas são, ou seja, o quão se aproximam da realidade. Primeiramente foi divido a base em treino (70%) e teste (30%). Dessa forma, foi possível calcular a acurácia das recomendações utilizando como comparação itens que já foram avaliados pelos usuários. A fim de calcular a acurácia das recomendações utilizamos a métrica RMSE (Root mean squared error). Essa métrica nos retorna a raiz do quadrado das diferenças entre o valor estimado e o valor real. Valor esse que pode ser obtido através da fórmula abaixo:

$$\text{RMSE} = \sqrt{\frac{\sum \left(P_{\text{est}} - P_{\text{true}}\right)^2}{n}} \tag{1}$$

Na (Tabela 3) encontram os resultados obtidos através da métrica RMSE apresentada acima. Observa-se que o valor obtido quando utilizado Train/Test foi diferente do valor obtido na submissão Kaggle, isso provavelmente ocorreu pelo fato de que quando utilizando parte do arquivo ratings.csv como test, uma grande parte dos itens viraram cold-start problem levando o score a abaixar. Um outro ponto a se observar, foi a estimativa utilizando Rocchio, apesar de ser um modelo complexo, para esse cenário obteve resultados relativamente ruins e muito inferiores a apenas utilizando a média imdb, esses resultados só podiam ser obtidos para amostras pequenas, para amostras grandes não estava escalando. Esses resultados podem ser observados na tabela abaixo.

Tabela 3. Scores obtidos através da métrica RMSE

Estratégia	Score
Average ImdbRating(BestScore Kaggle)	1.59062
Average imdbRating (train/teste)	2.2839
Rocchio (”Genre”)	–

6. Conclusão

Podemos concluir a partir dos resultados obtidos sobre essa base de dados que o cálculo das estimativas utilizando os imdb ratings foi superior a estratégia de recomendação utilizando Rocchio. Esse resultado pode estar enviesado pelo fato de que 23% dos itens targets são cold start e de que a matriz de token era muito esparsa, o que fazia com que a estimativa do rocchio ficasse baixa. Obtivemos um tempo médio de execução de 15s e um score obtido com o RMSE de 1.59062, valores que podem ser melhorados em trabalhos futuros.