Relatório - Trabalho Prático 2

Isadora Alves de Salles
Matrícula: 2017014553
Universidade Federal de Minas Gerais

1 INTRODUÇÃO

O objetivo deste trabalho foi implementar uma solução para uma recomendação de filmes baseada em conteúdo. Várias escolhas de implementação afetam a recomendação, nesse trabalho o foco maior foi sobre a escolha da representação do conteúdo, quais categorias dos conteúdos utilizar, como seria feito o cálculo de similaridades e qual o peso dado a cada categoria do conteúdo.

2 IMPLEMENTAÇÃO

2.1 Pré-processamento do conteúdo

O arquivo de conteúdo, passado como entrada para o programa, possui em cada linha o identificador do item e o respectivo conteúdo do item, que é um dicionário que pode conter as seguintes categorias: 'seriesID', 'Writer', 'Runtime', 'Language', 'Metascore', 'Title', 'Poster', 'Season', 'Director', 'imdbID', 'Response', 'imdbVotes', 'Episode', 'Year', 'Rated', 'Genre', 'Released', 'Country', 'Actors', 'imdbRating', 'Type', 'Awards', 'Error'. A leitura do arquivo de conteúdo foi auxiliada pelo uso da biblioteca Json, nativa do Python, para transformar a *string* com os conteúdos em um dicionário. Então, foi criado um dicionário em que a chave é o identificador do item, e os valores serão as categorias do conteúdo desses itens pré-processadas.

O pré-processamento do conteúdo dos itens foi feito separadamente por cada categoria. Para as categorias 'Plot', 'Genre', 'Title', 'Language' e 'Country' apenas realizei uma separação das palavras em *tokens*, utilizando o espaço como separador, ou seja, tratei as palavras como *unigrams*. Já para as categorias 'Director', 'Actors' e 'Writer', os dados foram transformados em *tokens* utilizando a vírgula como separador, de forma que teremos *unigrams* representando os nomes dos diretores, atores e escritores. Nesses dois casos foram excluídas as pontuações, as palavras insignificantes (*stopwords*) e todas as letras das palavras foram transformadas para minúsculas. Porém, percebi que em alguns casos das categorias de nomes (diretor, atores e escritor), além do nome existia também uma breve descrição da pessoa entre parênteses, por exemplo: "Tony Gatlif (screenplay)". Por conta disso, nesse caso também exclui todas as palavras que aparecem entre parênteses.

Foram tratadas também duas categorias numéricas: 'Year' filtrado para armazenar apenas a década em que o filme foi lançado, isso foi feito retirando o último dígito do ano; e 'Runtime' filtrado para também retirar o último dígito do número de minutos que um filme possui, assim teríamos um enfoque nas dezenas de minutos.

Após realizar o pré-processamento de cada categoria, foi criada uma representação vetorial para cada categoria dos conteúdos separadamente utilizando o TF-IDF em todas elas:

$$TF(t) = \frac{\text{Frequency occurance of term t in document}}{\text{Total number of terms in document}}$$

$$IDF(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing term t}}\right)$$

$$TF - IDF(t) = TF(t) * IDF(t)$$

2.2 Realizando a Predição

A matriz de utilidades recebida como entrada para o programa é bastante esparsa, de forma que não é possível armazená-la na memória. Então, ela foi armazenada num formato de lista de adjacências, utilizando um dicionário, que é uma estrutura mais simples e nativa do Python, e portanto possui um tempo de acesso menor. Sendo assim, foi criado um dicionário com as chaves sendo os IDs dos usuários, e para cada usuário foi armazenado os IDs dos itens que ele avaliou e a respectiva nota dada.

Possuindo a representação vetorial para cada categoria pra cada item e as avaliações passadas dos usuários, foi possível então realizar as predições. O algoritmo utilizado para as predições segue a ideia de uma hibridização paralelizada utilizando um vetor de pesos para combinar a predição obtida por vários recomendadores. Todos os recomendadores utilizados são baseados em conteúdo, a diferença entre eles é apenas o tipo de conteúdo (a categoria) utilizado para fazer a predição. Cada uma dessas categorias que foram pré-processadas foram utilizadas em um recomendador separado, e as predições obtidas por cada recomendador foram combinadas. A escolha dos melhores pesos e das categorias a serem utilizadas (dentre as pré-processadas) será discutida na próxima sessão.

O *score* obtido por cada recomendador foi computado utilizando a ideia de uma predição baseada em item. Foi computada a similaridade do cosseno item-item, entre o item alvo e os itens consumidos pelo usuário:

$$s(a,b) = \cos(a,b) = \frac{ab}{\|a\|\|b\|} = \frac{\sum_{i=1}^{d} a_i b_i}{\sqrt{\sum_{i=1}^{d} a_i^2}\sqrt{\sum_{i=1}^{d} b_i^2}} \quad (1)$$

Depois essas similaridades calculadas foram utilizadas para realizar uma média ponderada das avaliações passadas do usuário, isso dará a nota predita para a categoria avaliada:

$$\widehat{r}_{ui} = \frac{\sum_{j=1}^{n} s(a,b) r_{uj}}{\sum_{j=1}^{n} s(a,b)} \quad (2)$$

Após calcular a predição baseada em item para cada categoria de conteúdo utilizada, essas predições foram agregadas utilizando pesos para encontrar a predição final para o item alvo.

Também foi necessário definir quais estratégias tomar para lidar com *cold-start*. Por estar utilizando vários recomendadores separados, um para cada categoria, podem existir itens que não possuem a categoria avaliada. Um outro caso, é quando, em alguma categoria, o item é completamente diferente dos itens já consumidos pelo usuário. Nesses dois casos, não faz sentido dizer que o usuário daria nota zero para o item, então esses casos foram tratados como *cold-start* de item e foi seguida uma estratégia de imputação, utilizando o 'imdbRating' do item, e caso o item não possua esse valor, foi imputado a média das avaliações do usuário. Analogamente, quando o *cold-start* é de usuário, foi utilizado novamente o 'imdbRating' do item, e caso o item não possua esse valor, foi imputado a média dos valores de 'imdbRating' dos itens da base.

2.3 Complexidade Computacional

A complexidade temporal do algoritmo citado recai sobre a etapa de cálculo do TF-IDF e sobre a computação das similaridades item-item. No caso do TF-IDF, para computar tanto o TF quanto o IDF, é necessário visitar todos os itens e percorrer todos os termos do documento já pré-processado. Na implementação, o IDF foi pré-computado. Dessa forma, a complexidade do TF-IDF é $O(nt)$, onde n é o número de itens e t é o número de termos. Já o custo do cálculo das similaridades item-item no pior caso será $O(n^2m)$, onde n é o número de itens e m é o número de usuários para os quais queremos recomendar um item.

Já a complexidade espacial recai sobre o armazenamento do TF-IDF para cada categoria de cada item, isso é dado por $O(nCt)$, onde n é o número de itens, C é a quantidade de categorias do conteúdo dos itens e t é a quantidade de termos em cada categoria.

3 EXPERIMENTOS

3.1 Ambiente computacional

Os experimentos foram executados em um computador de 12Gb de RAM, intel core i7 com 8 núcleos num sistema operacional Ubuntu 20.04LTS. O código leva cerca de 1 minuto para executar.

3.2 Discussão dos Resultados

Primeiramente, é necessário explicar como foi feita a escolha da imputação em casos de *cold-start* de item. Como dito anteriormente, quando o item não possui uma determinada categoria que está sendo avaliada ou o usuário ainda não consumiu nenhum item similar a este, consideramos que o item é um *cold-start* na categoria específica. Foram testadas 3 estratégias de imputação nesses casos, descritas na Tabela 1.

Estratégia de imputação	RMSE mínimo
Média das avaliações do usuário	1.71308
(Média das avaliações do usuário + 'imdbRating' do item alvo)/2	1.58985
'imdbRating' do item alvo	1.55096

Tabela 1: Estratégias de imputação testadas

Na Tabela 1 temos as estratégias de imputação testadas e o respectivo RMSE mínimo alcançado em cada uma delas. Note que adicionar o 'imdbRating' trouxe melhorias significativas para o resultado, e isso faz bastante sentido pois o 'imdbRating' trás uma noção de qualidade do filme. Então, mesmo que o filme não tenha características similares à filmes que o usuário já viu, isso não significa que o usuário irá odiar o filme. Certamente, a nota dada pelo usuário estará mais próxima do valor médio das avaliações do filme ('imdbRating') do que de zero. Utilizar essa característica presente no conteúdo dos itens tem um potencial muito grande nas predições baseadas em conteúdo. Ao invés de recomendarmos apenas itens muito similares aos itens que o usuário já viu no passado, temos a possibilidade de fazer com que filmes considerados bons pela crítica em geral, mas que não são o tipo de filme que o usuário costuma consumir, cheguem ao usuário. Ou seja, adicionar o 'imdbRating' na imputação ajuda a aumentar a diversidade da recomendação. Por conta disso, o recomendador é capaz de generalizar, e não se ater tanto apenas aos dados presentes nos dados de treino.

Além dessa decisão, foram feitas outras escolhas que levaram a diminuição do valor do RMSE nos dados de teste. Inicialmente, para começar a executar o algoritmo é necessário definir quais categorias de conteúdo serão utilizadas para a predição e quais os respectivos pesos dados para essas categorias. As categorias que podem ser utilizadas pelo algoritmo implementado são: 'Plot', 'Genre', 'Title', 'Language', 'Country', 'Director', 'Actors', 'Writer', 'Year' e 'Runtime'. Os testes sobre quais pesos e quais categorias funcionam melhor foram avaliados através do resultado expresso no *Kaggle*. Nas primeiras submissões utilizei apenas o 'Plot', ao agregar mais categorias notei uma queda considerável no RMSE. Porém, as categorias 'Language', 'Country' e 'Runtime' resultaram na piora da predição.

A adição do ano trouxe melhorias significativas para o resultado. Devido a isso, tentei usar uma outra abordagem, em que filmes que foram lançados no final da década "participariam" tanto da década em que foi lançado quanto da década seguinte, analogamente para filmes lançados no início da década. Essa abordagem resultou num erro maior no *Kaggle*, e por isso mantive a análise apenas por décadas. O gênero também foi uma categoria muito importante para a predição. E tanto o gênero quando o ano possuem poucos casos de *cold-start* na base. A adição de diretor, atores e escritor também melhorou a predição, e isso faz bastante sentido na prática, pois as pessoas costumam gostar de ver filmes produzidos por pessoas que elas gostam.

Os pesos foram ajustados a medida que foram observadas modificações no valor do RMSE. A combinação de categorias e seus pesos que resultou o melhor RMSE de teste (1.55096), foi:

Categoria	Pesos
Genre	3
Director	4
Year	5
Actors	3
Writer	3
Title	4

Tabela 2: Partes do conteúdo utilizadas

É importante mencionar que também foi testada a implementação do algoritmo Rocchio, criando uma representação vetorial para o usuário, agregando os itens que ele consumiu ponderados pela nota dada. Porém, as notas preditas por essa heurística não foram boas. Acredito que, podem existir usuários na base que consumiram muitos itens distintos fazendo com que a representação do usuário não seja tão significativa, e que apresente sempre uma similaridade muito baixa com os novos itens a serem recomendados.

4 DESAFIOS E CONCLUSÕES

O principal desafio do trabalho foi encontrar a escolha de pesos ideias para ponderar cada categoria de conteúdo a fim de diminuir o RMSE. Foram feitas várias tentativas no *Kaggle* com o propósito de entender o que mais influenciava na predição.

Com esse trabalho foi possível entender as dificuldades de se implementar um bom recomendador baseado em conteúdo. Com essa implementação foi possível atingir um resultado muito bom no RMSE de teste, acredito que o uso do 'imdbRating' em conjunto com a recomendação personalizada tem um papel bastante importante nisso, pois utilizando esse conhecimento sobre o item foi possível obter um RMSE até melhor do que o obtido no trabalho prático 1.