PL4 - Algoritmos Probabilísticos

Licenciatura em Engenharia Informática

Métodos Probabilísticos para Engenharia Informática

(2022-2023)

Turma P7

Docentes:

Professor Carlos Bastos

Professor António Teixeira

Alunos:

Bárbara Nóbrega Galiza – 105937

João Miguel Dias Andrade - 107969

Janeiro 2022

Conteúdo

1 Introdução

Esse trabalho tem como objetivo a compreensão dos mecanismos associados à similaridade através da criação de um menu interativo que dá sugestões ao utilizador baseado no que ele introduzir. Neste programa temos informação sobre diversos utilizadores, como o seu nome, interesses, filmes avaliados e os valores dessas avaliações. Para além disso temos também informação sobre cada filme individual como o seu titulo. O objetivo deste trabalho é utilizar esses valores para conseguir dar boas sugestões ao utilizador do que assistir e sugestões de outros utilizadores que seriam recomendados para avaliar o filme escolhido.

Neste relatório, será descrita a solução implementada, que foi dividida em dois scripts, "scriptOne.m"e "scriptTwo.m"que têm como função criar as estruturas de dados necessários e utilizar estas estruturas para obter os resultados respetivamente

2 Abordagem

Como indicado no enunciado, a solução foi dividida em dois scripts: um para o processamento de dados, e o outro para a interação com o utilizador através de um menu. No primeiro foram guardadas as informações dos ficheiros fornecidos e geradas as matrizes de assinaturas de suporte a cada opção. Já no segundo foi feita a deteção dos pares similares e algum processamento adicional necessário de acordo com a opção. Nesta secção, será explicada a abordagem que tivemos para processar os dados dos ficheiros de entrada, para criar o menu e para construir cada opção.

2.1 Processamento dos dados de entrada

Inicialmente, no scriptOne, usamos a função loadData(), disponibilizada na secção 4.3 do guião PL4, para extrair do ficheiro "u.data"um cell array contendo os IDs dos filmes avaliados por cada utilizador, o número total de utilizadores e um array com os IDs dos utilizadores. Para isso, fizemos algumas alterações na função original, a fim de que esta também adicionasse ao cell array Set a terceira coluna do ficheiro "u.data", correspondente à avaliação atribuída a cada filme.

Nessa função, utilizamos a instrução load() para salvar os dados de "u.data"em na variável udata. A partir dela, guardamos na variável u as colunas 1, 2 e 3 de udata, ficando assim com os IDs dos utilizadores, IDs dos filmes e avaliação dos filmes. Depois, obtemos o array users com os IDs dos utilizadores através da função unique() aplicada à primeira coluna de u, e o número de utilizadores fazendo length() desse array, que devolverá seu o número de linhas. Em seguida, criamos um cell array com o tamanho igual ao número de utilizadores, e percorremos os IDs dos utilizadores em um ciclo for. Dentro desse ciclo, utilizamos o find() para obter os índices em que o user dessa iteração aparece no array u. Assim, para cada utilizador, adicionamos uma linha (célula) no cell array Set com o conteúdo das colunas 2 e 3 do array u de cada índice devolvido pelo find(), ou seja, cada filme avaliado pelo utilizador.

Após chamar essa função, executamos duas vezes a instrução indicada no guião readcell(): a primeira para ler os valores presentes no ficheiro "users.txt"e separados pelo delimitador ";"para a variável dic e a segunda para ler os valores presentes no ficheiro "film_info.txt"e separados por tab (\t) para a variável movieTitles. Depois, optamos por juntar a informação obtida do ficheiro "users.txt"com a informação do ficheiro "u.data", para facilitar a manipulaçao futura. Para isso, utilizamos uma série de atribuições e indexações: primeiro, criamos o cell array dicInfoByUsers, com NumUsers

(= número de utilizadores) linhas e 18 colunas, pois a matriz dic possui 17 colunas e iremos adicionar mais uma correspondente a variável moviesByUser, que recebeu o Set devolvido pela função loadData(). Depois, adicionamos as três primeiras de dic colunas para dicInfoByUsers, e logo em seguida, adicionamos o vetor coluna moviesByUser. Por fim, adicionamos as colunas restantes, índices 4 a 17 de dic, às colunas 5 a 18 de dicInfoByUser. Assim, concluímos essa parte do problema.

2.2 Criação das estruturas de dados

No segundo script são necessárias algumas estruturas de dados que são criadas no primeiro script. As opções dois e três apenas precisam das assinaturas, a primeira opção não necessita de nada e a opção quatro necessita também de um Counting Bloom Filter.

Por isso, utilizamos a instrução save() para salvar tudo o que é necessário no ficheiro "data.mat", o qual demos load() no scriptTwo.

2.2.1 Opção 2

Para a opção dois é necessário calcular as assinaturas correspondentes aos conjunto de utilizadores que avaliaram cada filme. Para este fim criámos a função minHashUsersByMovies() que é baseada na implementação feita no PL4, sendo a única diferença que em vez de calcularmos as assinaturas para os filmes de cada utilizador, calculamos-las para os utilizadores que avaliaram cada filme. Como os filmes são representados por um ID que vai de 1 a 1682 não necessitamos de passar um array com esta informação, em vez disso, utilizamos a função getUsersByMovie() por cada filme que iteramos para obter a lista dos utilizadores que o avaliaram.

2.2.2 Opção 3

Nesta opção, tal como na opção anterior, apenas precisamos de calcular as assinaturas, o que é feito através da função minHashInterests que, tal como a minHashUsersByMovies() é uma modificação de uma implementação feita na aula prática. Neste caso em vez de se criar as assinaturas baseadas nos filmes assitidos, criamos as assinaturas baseadas nos interesses de cada utilizador,

2.2.3 Opção 4

A opção quatro é a que requer calcular criar mais estruturas de dados, pois para além das assinaturas necessitamos também de criar um Counting

Bloom Filter e de dividir os títulos em shingles. Este Bloom Filter será utilizado para armazenar o número de avaliações com nota superior a três para cada filme.

Em primeiro lugar criamos o Bloom Filter. Para criação e maneusamento do Bloom Filter nós adicinámos três funções:

- bloom_filter_initialize() - inicializa o Bloom Filter
- bloom_filter_increment() - incrementa o valor de um elemento por um
- bloom_filter_get_element() - obtém o valor de um elemento (fazendo o minimo de todos o que encontra)

Seguidamente, iremos popular o Bloom Filter, iterando por todos os users e por todos os filmes que estes avaliaram e adicionando, caso tenham uma avaliação superior a três, usando a função bloom_filter_increment().

A última estrutura necessária é a que contém as assinaturas dos títulos mas, para as podermos calcular, teremos de primeiro dividir cada titulo em diversos shingles. Podemos obter o set de shingles iterando por todos os titulos e seguidamente por todas as substrings desses títulos com o tamanho dos shingles. Esse set de shingles é então usado pela função minHashTitles() para calcular as assinaturas dos shingles dos titulos. A função minHashTitles() é também uma variação da função minHash() implementada no PL4 que, neste caso, calcula as assinaturas para cada shingle de cada titulo.

2.3 Menu

Antes de criar o menu, foi preciso inserir código para pedir o Id do filme ao utilizador e para verificar se o mesmo é válido. Para isso, inicializamos a variável fimId a 0 e criamos um ciclo while com a condição de id inválido, *i.e.*, id abaixo de 1 ou maior que 1682. Dentro desse while, adicionamos o input para que ele fosse impresso até que o utilizador inserisse um id válido, além de um if para imprimir uma mensagem de erro.

Já ara fazer o menu, utilizamos a função menu() do MATLAB que permite ao utilizador escolher uma das opções especificadas por nós através de uma caixa de diálogo. Para que o utilizador possa escolher mais de uma opção em uma só "run", essa função foi posta dentro de um while true, sendo que a opção 5, de saída, dá break ao ciclo. Para processar cada opção foi usado um switch case com o valor devolvido por menu().

Todo o código descrito nessa secção foi inserido no scriptTwo.

2.4 Opção 1

A opção 1 tem como objetivo a listagem dos IDs e nome dos utilizadores que avaliaram o filme actual. Para obter esse resultado, optamos por criar uma função getUsersByMovie(), cujos argumentos são o dicInfoByUser, o NumUsers e o filmId inserido pelo utilizador, e o retorno um cell array, usersByMovie, com número de linhas correspondente ao número de utilizadores que avaliaram o filme e 3 colunas: a primeira com o ID do utilizador, a segunda com o nome e a terceira com o apelido.

No corpo dessa função, inicialiamos usersByMovie como um cell array vazio e guardamos a coluna 4 de dicInfoByUser na variável Set. Depois, percorremos em um ciclo for os IDs dos utilizadores, e para cada id, guardamos na variável movie os valores devolvidos ao indexar com chavetas o Set na posição "id"na coluna 1, ou seja, os filmes desse utilizador. Em seguida, usamos um if e um find() para testar se o array booleano resultante da comparação entre movies e filmId tinha algum campo a 1, ou seja, se o filmId estava presente na lista de filmes desse utilizador. Caso verdade, adicionamos à última linha de usersByMovie o ID do utilizador, seu nome e seu apelido, presentes nas colunas 2 e 3 do dicInfoByUser.

Optamos por chamar essa função antes do switch no scriptTwo, pois a variável dicInfoByUser também vai ser usada no processamento de outras opções.

2.5 Opção 2

A segunda opção é composta por dois passos: obter os dois filmes mais similares (baseado nos utilizadores que avaliaram cada filme) ao filme inserido pelo utilizador e apresentar uma lista de utilizadores que tenha avaliado pelo menos um dos filmes similares mas que não tenham avaliado o filme selecionado.

Primeiramente, para calcular os filmes similares, criámos a função detectSimilarMoviesByUsers() que calcula a distância entre um filme e todos os outros filmes cujas assinaturas são passadas como argumentos. Isto é possível utilizando uma versão modificada da implementação das distancia de Jaccard usando assinaturas que fizemos na secção 4.3 do guião PL4. Nesta versão apenas calculamos a distância de um certo filme aos outros todos pois, para este caso não necessitamos de mais. Depois de obtermos esta lista de distâncias retiramos os dois filmes mais similares.

Para obter os utilizadores iremos utilizar a função getUsersByMovies(). Primeiramente iremos iterar pelos dois filmes mais similares e, para cada um, chamar a função getUsersByMovies() passando-os como argumentos.

Seguidamente iremos iterar por todos os utilizadores obtidos e verificar se:

- viram o filme da iteração atual
- não viram o filme selecionado pelo utilizador
- ainda não "passámos"por esse utilizador

No final da verificação nós guardamos no array *visited* na posição correspondente ao ID do utilizador o valor 1, que depois usamos na última parte da verificação para não repetirmos utilizadores. Caso as três condições sejam cumpridas damos print ao ID e nome do utilizador.

2.6 Opção 3

A opção 3 tem como objetivo a sugestão de utilizadores a partir de conjuntos similares de interesses. Mais especificamente, pretende selecionar utilizadores que ainda não tenham avaliado o filme atual e cuja distância de Jaccard aos utilizadores que já avaliaram seja inferior a 0.9. Além disso, deve devolver os IDs e nomes dos utilizadores que aparecem no maior número de conjuntos, em que cada conjunto corresponde à lista de utilizadores que ainda não avaliaram para cada utilizador que avaliou.

Para implementá-la, primeiro criámos a função minHashInterests, que aplica o algoritmo MinHash e devolve a matriz de assinaturas sigInterests correspondente aos conjuntos de interesses dos utilizadores. Chamamos essa função no scriptOne lhe passando como argumento as variáveis dicInfoByUser, numUsers e numHash (número de hash functions). A escolha do valor de numHash irá ser explicada em uma secção a parte a seguir.

Dentro da função, começamos por inicializar a matriz sigInterests com o valor Inf existente no MATLAB, e que corresponde ao "infinito". Isso foi feito porque irámos precisar do mínimo entre os valores da matriz para calcular o minHash. Além disso, sua dimensão foi especificada como NumUsers linhas e numHash colunas. Depois, inicializamos a waitbar para acompanhar o progresso e criamos um for para percorrer todos os ids dos utilizadores. Para cada utilizador, guardamos na variável interestList as colunas 5 à 17 do dicInfoByUser na linha n1, sendo n1 o id do utilizador atual. Em seguida, para remover os campos "missing"presentes em algumas linhas, incializamos um array vazio interests, e criamos um ciclo for que itera de 1 a 14 e guarda o valor na variável int. Essa variável corresponde ao índice do interesse do utilizador presente no interestList. Dentro do ciclo, testamos se o tamanho do interesse é maior que 1, visto que os campos "missing"sempre terão tamanho 1 e as strings sempre terão tamanho maior que 1. Caso verdade, adicionamos o interesse ao array interests.

Após remover esses campos, procedemos para a aplicação do algoritmo MinHash. Para isso, guardamos na variável Ninterests o tamanho de interests, ou seja, o número de interesses do utilizador nessa iteração, e criamos um ciclo for para percorrer cada interesse, guardando em i o valor de 1 a Ninterests. Nesse ciclo, guardamos em key o valor de interests na posição i, e o inserimos como argumento na chamada à função DJB31MA_multiple, junto com a seed e o número de hash functions (mais a frente será explicada à alteração à função DJB31MA que originou essa função). Essa retorna um array, h_out, com numHash elementos correspondentes aos hashcodes devolvidos por cada hash function. Em seguida, adicionamos à linha n1 de sigInterests o mínimo entre h_out e a mesma linha em que iremos adicionar, para assim obter os menores hash codes entre todos os conjuntos de interesses. Assim, deletamos a waitbar e terminamos a função.

Já com a matriz de assinaturas, procedemos com a deteção dos pares similares, utilizando a função detectSimilarsByInterest. Essa função é chamada no scriptTwo, e antes de ser chamada fazemos um ciclo for para obter apenas os ids dos utilizadores que já viram o filme, usando os dados que tinhamos guardado na variável usersByMovie. Insermos esses IDs como argumento da função, em conjunto com a matriz sigInterests, o numHash, os users (lista de todos os utilizadores) e um id de um utilizador que já viu o filme, que vai ser obtido através da iteração de um ciclo for, no qual esta função está encapsulada, de 1 a numUsersByMovie, que é o tamanho de usersByMovie.

Dentro dessa função, definimos o threshold para 0.9, e guardamos em uma variável usersRecommended o conjunto resultante da diferença entre o users e o usersAlreadyWatched, que recebeu o valor de usersByMovie, através da função setdiff(). Assim, obtemos todos os utilizadores que ainda não avaliaram o filme. Inicializamos o counter c a 1, e criamos um ciclo for para percorrer os usersRecommended. Nesse ciclo, obtivemos a distância de Jaccard fazendo 1 - número de assinaturas iguais entre o userRecommended e e o user de ID enviado como argumento divido pelo número de hash functions. Em seguida, fazemos um if para testar se o id do user da iteração atual é diferente do id recebido como argumento e se a distancia é menor que o threshold. Caso verdade, adicionamos ao cell array de retorno, similarUsers no índice c e incrementamos o contador.

Por fim, já com os utilizadores similares, fazemos um array user_appear_count para guardar o número de aparições de um user. Através de dois ciclos for, percorremos os utilizadores e verificamos se o id desse utilizador está presente no cell array de similare, através da função ismember(). Caso verdade, adicionamos em user_appear_count no id correspondente a contagem de aparições. Depois, fazemos sortrows() para obtermos os ids dos utilizadores que mais aparecem, e imprimos seus dados no terminal.

2.7 Opção 4

Na opção 4 o objetivo é obter os 3 nomes de filmes mais similares a uma string dada pelo utilizador. O primeiro passo é então pedir ao utilizador para escrever o titulo, ou parte do titulo de um filme. Seguidamente temos de dividir a string fornecida em shingles, tal como fizemos com os títulos no primeiro script. Com estes shingles podemos calcular então as assinaturas para os shingles desta string usando a função minHashTitles().

Depois de termos as assinaturas podemos calcular as distâncias entre a string fornecida e os títulos dos filmes, usando a função detectSimilarsByString() que tal como a função detectSimilarMoviesByUsers() é uma versão alterada da implementação do guião PL4 da distancia de Jaccard usando assinaturas. Neste caso, esta função, usa as assinaturas dos shingles de uma dada string e as assinaturas dos shingles dos títulos para calcular os nomes de filmes mais próximos á string dada.

Assim com as distancias dos títulos já calculadas podemos ordenar,em ordem das distâncias, o array onde estas estão guardadas e iterar pelas 3 primeira linhas, dando print ao ID do filme, o seu nome e a quantidade de avaliações superiores a 3 que podemos obter usando a função bloom_filter_get_element().

2.8 Opções Tomadas na Implementação

Nas nossas implementações optamos por usar um número de hash functions para os minHashs igual a 200, por concluir durante a realização do guião 4.3 que esse era o número necessário para que não houvessem erros. Além disso, escolhemos 6 como o número de hash functions k do bloom filter devido à aula teórica e o valor determinado como k ótimo. Para o tamanho n do bloom filter, também usamos o cálculo do valor teórico e concluímos que 16000 garantia uma probabilidade de falsos positivos de aproximadamente 0.01. Para o tamanho dos shingles, testamos com os valores de 2 a 5 e concluímos que com 3, mais associações corretas eram feitas.

Acerca da função DJB31MA, a modificamos para permitir que essa devolvesse logo um array com 200 hashcodes, o que nos permitiu usar esse número variável de hash functions.

3 Conclusão

A realização deste trabalho permitiu-nos aprofundar os nossos conhecimentos sobre similaridades, o que é um tópico bastante importante hoje em dia, já que, por exemplo, todos os websites pretendem obter os melhores resultados do que mostrar aos seus utilizadores para os manter interessados.

Para a sua realização, foi necessário o conhecimento prévio obtido nas aulas práticas e teóricas não só sobre a similaridade mas também sobre minHash e Filtros de Bloom. Um dos principais desafios foi a adaptação dos programas que implementamos nas aulas práticas para poderem ser usados nos contextos pedidos neste enunciado. Após estudar minuciosamente cada detalhe, foi mais fácil fazer estas alterações que nos permitiram chegar ao resultado final.