universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

PL 4 – Ano Letivo 2022/2023

Algoritmos Probabilísticos

Métodos Probabilísticos para Engenharia Informática - P7

Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

José Mendes - 107188
Filipe Obrist – 107471

Índice

Introdução

Neste trabalho, no âmbito da disciplina de Métodos Probabilísticos para Engenharia Informática, pretendemos desenvolver, em Matlab, uma aplicação com algumas funcionalidades de um sistema online de disponibilização de filmes.

De modo a verificar as condições referidas no enunciado do problema, é necessário o uso e compreensão de várias ferramentas trabalhadas no guião 4 desta cadeira. Para a realização deste projeto, este foi subdividido em duas grandes partes: **data** (ler os ficheiros de entrada e guardar as estruturas de dados associadas aos utilizadores e filmes) e a **main** (ler do disco todas as estruturas previamente guardadas pelo **data** e implementa as interações com o utilizador).

Neste relatório estão explicadas as várias fases de desenvolvimento do código, tal como a escolha do método de desenvolvimento para cada opção pedida no enunciado e o papel de cada função envolvida neste programa.

Assim, explicaremos individualmente o que foi pensado para cada opção de interação que o utilizador pode selecionar.

Opção 1

A aplicação lista os nomes dos utilizadores que avaliaram o filme atual

Inicialmente, o script **data** começa por ler as três primeiras colunas do ficheiro **u.data**, que possui 100000 avaliações realizadas a 1682 filmes por 943 utilizadores. A primeira coluna corresponde aos utilizadores, a segunda aos filmes e a terceira à sua avaliação. De seguida, neste mesmo script é lido, também, o ficheiro **users.txt**, que guarda o *id* dos *users*, bem como o seu nome e os seus interesses, tudo isto separado por ";".

Para o desenvolvimento desta opção, foi criada, no script **data**, a função **user_review**. Esta função vai devolver o **rated_users**, que será um *cell array* de dimensões 1682x2 (1682 pois são 1682 filmes) em que para cada filme irá guardar os nomes completos e os ids de cada utilizador que já o avaliou. Assim, o nome e sobrenome do utilizador foi-se buscar ao *cell array* **dic** (nomes na segunda coluna e sobrenomes na terceira) e os *ids* tanto como dos filmes como dos *users* são obtidos através da primeira e segunda coluna da variável **u**, respetivamente.

```
%% option 1
rated_users = user_review(u, dic);
```

```
function rated_users = user_review(u, dic)
    rated_users = cell(1682, 2); % pois existem 1682 films

        for i = 1:length(u) % vai percorrer a lista de avaliações para ver se o film esta la
            id = u(i,2);
            user_id = u(i,1);
            name = string(dic{user_id, 2}); % o nome do utilizador
            surname = string(dic{user_id, 3}); % sobrenome do utilizador
            fullname = name + " " + surname;
            rated_users{id,1} = [rated_users{id,1} fullname];
            rated_users{id,2} = [rated_users{id,2} user_id];
        end
end
```

Tendo então este *cell array*, agora no script **main** é só necessário ir buscar a informação pretendida para o filme atual. Utilizando um *loop for* para percorrer cada um dos *users*.

```
switch inp
    case 1
        names = rated_users{film_id,1};
        ids = rated_users{film_id,2};
        fprintf("Users that rated this movie:\n");
        for i = 1:length(names)
            fprintf("User ID: %d; Name: %s\n", ids(i), names(i));
        end
```

Opção 2

A aplicação determina os dois filmes mais similares ao atual e lista os utilizadores que avaliaram pelo menos um destes filmes, mas não o atual

Nesta interação, foi necessário desenvolver duas funções no script **data**. A primeira função, **list_of_users**, é criada com o objetivo de criar um *cell array* (**users_per_film**) onde se guarda os utilizadores que avaliaram cada filme (mecanismo e pensamento muito parecido ao da função **user_review** da opção 1).

```
%% option 2
n_hash_functions = 100;
users_per_film = list_of_users(u);
similar_films = calculate_film_similarities(n_hash_functions, users_per_film);
```

```
function users_per_film = list_of_users(u)
    users_per_film = cell(1682, 1); % n de filmes em que cada linha é um user
    for i = 1:length(u)
        % os indices da matriz linha representam os ids do filme
        film = u(i,2);
        user = u(i,1);
        users_per_film{film} = [users_per_film{film} user]; % para adicionar o user à lista do filme
    end

end
```

De seguida, é desenvolvida a função **calculate_film_similarities**, que faz uso ao *cell array* obtido na função anterior. Nesta função serão calculados os MinHash entre os filmes. Para tal, é desenvolvido um ciclo *for*, que percorre todos os filmes, com o intuito de obter a matriz MinHash e a matriz com as similaridades entre filmes. Dentro deste ciclo, começa-se por percorrer os *users* que já assistiram a esse determinado filme usando outro *for*. Neste, são então aplicadas as funções de dispersão para obter as assinaturas referentes a todos os utilizadores que esse filme possui. Antes de avançar para o próximo filme, guarda as assinaturas referentes ao filme na sua linha (o *id* do filme corresponde à sua linha).

Posteriormente, é, então, necessário calcular a similaridade entre todos os filmes. Com este objetivo, é então realizado um duplo *for* que permite realizar as comparações entre filmes sem repetição. Para a comparação é então utilizada a matriz de assinaturas obtida anteriormente, e, uma vez que cada linha desta matriz é referente a um filme, basta comparar as linhas de cada filme. Caso sejam iguais, obtemos uma distância de 0 e caso sejam completamente diferentes obtemos uma distância de 1. O cálculo da distância é efetuado, igualando as linhas para obter uma matriz de zeros e uns, somando essa matriz, dividindo-a pelo número de *hash functions* aplicadas e, por fim, subtraindo por um.

```
function similar_films = calculate_film_similarities(n_hash_functions, users_per_film)
    minhash_films = inf(1682, n_hash_functions); % matriz que vai receber os resultados minhash
    h = waitbar(0,'Calculating...');
    for i = 1:length(users_per_film)
        waitbar(i/length(users_per_film),h);
        users = users_per_film{i}; % users que avaliaram o filme
        for j = 1:length(users) % cada user
            user = users(j);
            indexes = zeros(1, n_hash_functions);
            for z = 1:n_hash_functions
                user = [user num2str(z)];
                indexes(z) = DJB31MA(user, 181); % numero primo alto
            end
            minhash_films(i,:) = min([minhash_films(i, :) ; indexes]);
        end
    end
    delete(h);

    % de seguida é necessario calcular as similaridades
    similar_films = zeros(length(users_per_film));
    h = waitbar(0,'Filtering...');
    for film_1 = 1:length(users_per_film)
        waitbar(film_1/length(users_per_film),h);
        for film_2 = film_1+1:length(users_per_film)
            similar_films(film_1, film_2) = 1 -( sum(minhash_films(film_1, :) == minhash_films(film_2, :)) / n_hash_functions);
        end
    end
    delete(h);
end
```

Agora, já na **main**, é necessário procurar na **similar_films** os dois filmes que sejam mais similares ao filme atual. Para tal, percorremos a matriz primeiro para os filmes com *id* menor que o do filme atual e depois para os com *id* maior. À medida que se vai lendo a matriz, fica guardado a distância mínima entre filmes e os dois filmes mais similares até aquele momento.

```
case 2
    % como ja temos a matriz com as similaridades, temos agora de
    % ver os dois filmes mais similares
    % para obter as duas menores distancias
    min_distance = 1;
    simi_film_1_id = 0;
    simi_film_2_id = 0; % guarda os ids dos filmes

    % os filmes anteriores estao na coluna e os a seguir em linha
    % para os filmes com um id a baixo:
    for i = 1:film_id-1
        if similar_films(i, film_id) <= min_distance
            simi_film_2_id = simi_film_1_id; % guarda o valor anterior logo o segundo menor
            min_distance = similar_films(i, film_id);
            simi_film_1_id = i;
        end
    end

    % para os filmes com id a cima:
    for i = film_id+1:length(similar_films)
        if similar_films(film_id, i) <= min_distance
            simi_film_2_id = simi_film_1_id;
            min_distance = similar_films(film_id, i);
            simi_film_1_id = i;
        end
    end
```

Por fim, investiga-se os utilizadores possíveis para avaliarem o filme atual. Para encontrá-los, foi armazenado no **possible_users** os **users** que avaliaram os dois filmes mais similares e de seguida verificou-se se estes já tinham ou não avaliado o filme em causa. Caso ainda não tivessem avaliado o filme introduzido, são, portanto, imprimidos para o ecrã, com o seu *id* e nome.

```
possible_users = [];

first_id_users = users_per_film{simi_film_1_id, 1};
for i = 1:length(users_per_film{simi_film_1_id, 1})
    possible_users = [possible_users first_id_users(i)];
end

second_id_users = users_per_film{simi_film_2_id, 1};
for i = 1:length(users_per_film{simi_film_2_id, 1})
    if ~any(possible_users == second_id_users(i)) % caso ainda nao esteja no conjunto
        possible_users = [possible_users second_id_users(i)];
    end
end

fprintf("Possible users:\n");
for j = 1:length(possible_users)
    if ~any(users_per_film{film_id,1} == possible_users(j)) % caso nao pertença ao conjunto do filme original
        id = possible_users(j);
        name = dic{id,2};
        surname = dic{id,3};
        fullname = name + " " + surname;
        fprintf("User ID: %d; Name: %s\n", id, fullname);
    end
end
```

Opção 3

Para cada avaliador do filme atual, a aplicação seleciona dois utilizadores sugeridos que ainda não avaliaram o filme

O início da implementação desta opção é quase idêntico ao da opção anterior. Começa-se, portanto, por criar a função **list_of_interests** que irá devolver um *cell array* com os interesses de cada *user*. Isto é realizado percorrendo **dic**, que possui, para cada user, os seus interesses. Tem início na coluna 4 (é aí que começam os interesses) e até acabarem, isto é, até estar <missing>. Cada linha do criado *cell array* representa um *user* e o seu conteúdo é, por sua vez, os seus interesses.

```
%% option 3
n_hash_functions = 100;
interests_per_user = interests(dic); % obtem os interesses de cada user
similar_interests = calculate_interest_similatities(n_hash_functions, interests_per_user);
```

```
function interests_per_user = list_of_interests(dic)
    interests_per_user = cell(length(dic), 1);
    for i = 1:length(dic)
        interests = [];
        for j = 4:width(dic)
            interest = dic{i, j};
            if ismissing(interest)
                break;
            end
            interests = [interests string(interest)];
        end
        interests_per_user{i,1} = interests;
    end
end
```

Seguidamente, é criada a função **calculate_interest_similatities**, que sendo bastante semelhante à da opção anterior, tem também o objetivo de obter a matriz com as similaridades. Desta vez entre conjuntos de interesses.

Tal como a função previamente apresentada, **calculate_film_similarities**, começa por percorrer todos os *users* pelo seu conjunto de interesses utilizando um duplo *for*, o primeiro para percorrer os utilizadores e o segundo os interesses de cada um. Aplica as *hash functions* em cada interesse desse *user*, e após percorrer todos, guarda o resultado obtido na variável **minhash_interests**. Por fim, após estes ciclos for obtemos a matriz de MinHash para cada *user* utilizando os seus interesses.

A seguir, mais uma vez em semelhança à opção anterior é necessário calcular a distâncias de Jaccard entre todos os *users*. Com esse objetivo é iniciado um *for* que faz a comparação entre todos os filmes, sem repetição. A distância entre *users* é calculada igualando as linhas da matriz de MinHash, somando (obtendo assim o número de assinaturas iguais), dividindo pelo número de *hash functions* e subtraindo por um.

```
function similar_interests = calculate_interest_similatities(n_hash_functions, interests_per_user)
    minhash_interests = inf(length(interests_per_user), n_hash_functions); % matriz que vai receber os resultados minhash
    h = waitbar(0,'Calculating...');
    for i = 1:length(interests_per_user)
        waitbar(i/length(interests_per_user),h);
        interests = interests_per_user{i}; % users que avaliaram o filme
        for j = 1:length(interests) % cada user
            interest = char(interests(j));
            indexes = zeros(1, n_hash_functions);
            for z = 1:n_hash_functions
                interest = [interest num2str(z)];
                indexes(z) = DJB31MA(interest, 181); % numero primo alto
            end
            minhash_interests(i,:) = min([minhash_interests(i, :) ; indexes]);
        end
    end
    delete(h);

    % calcular as distancias
    similar_interests = zeros(length(interests_per_user));
    h = waitbar(0,'Filtering...');
    for interest_1 = 1:length(interests_per_user)
        waitbar(interest_1/length(interests_per_user),h);
        for interest_2 = interest_1+1:length(interests_per_user)
            similar_interests(interest_1, interest_2) = 1 -( sum(minhash_interests(interest_1, :) == minhash_interests(interest_2, :)) / n_hash_functions);
        end
    end
    delete(h);
end
```

No script **main**, uma vez que já possuímos as distâncias de Jaccard entre todos os *users*, apenas é necessário verificar, para o filme introduzido, os utilizadores que o assistiram, e com estes obter os que possuem uma distância inferior a 0.9 e que não tenham assistido o filme. Além disso, é necessário obter os dois *users* que aparecem mais vezes nesse conjunto e identifica-los.

Inicialmente, percorremos então os *users* que avaliaram o filme introduzido. Para cada *user* é necessário percorrer, então, a matriz com as distâncias de Jaccard, **similar_interests**. Primeiro os filmes com *id* menor (primeiro *for*) e de seguida os com *id* maior (segundo *for*), verificado aqueles que possuem uma distância menor que 0.9. Caso possua, vai verificar se esse *user* não avaliou o filme introduzido, tal é feito usando a negação da função **any**. Caso também se verifique é incrementado o *array* previamente declarado a zeros, **common_users**, em que a posição significa o *id* do user.

```
for k = 1:length(users)
    user = users(k);
    %filmes anteriores
    for i = 1:user-1
        similar_rate = similar_interests(i,user);
        if similar_rate <= 0.9
            continue;
        end
        % ver se ainda nao avaliou o filme
        if ~any(users_per_film{film_id} == i) % caso nao possua o filme introduzido pode adicionar
            common_users(i) = common_users(i) + 1;
        end
    end

    % filmes seguintes
    for i = user+1:length(similar_interests)
        similar_rate = similar_interests(user, i);
        if similar_rate <= 0.9
            continue;
        end
        if ~any(users_per_film{film_id} == i) % caso nao possua o filme introduzido pode adicionar
            common_users(i) = common_users(i) + 1;
        end
    end
end
```

Finalmente, é necessário ver quais são os dois user que aparecem mais nesse conjunto **common_users**. Para tal é utlizado um *if* em que caso o número de vezes que o *user* aparece for maior que um dos valores até então guardados, um destes passa a tomar um novo valor, alterando também o seu *id* para o deste novo *user*. Caso o maior valor seja alterado, ou seja, é encontrado um menor valor para este, as suas informações passam para as variavéis que guardam os valores do segundo maior valor. Por fim, basta obter o nome completo destes dois utilizadores e imprimi-los.

```
% temos de escolher os 2 users que aparecem mais vezes
first_user = 0;
id_1 = 0;
second_user = 0;
id_2 = 0;
for i = 1:length(common_users)
    n_times = common_users(i);
    if n_times > first_user
        second_user = first_user;
        id_2 = id_1;
        first_user = n_times;
        id_1 = i;
    elseif n_times > second_user
        second_user = n_times;
        id_2 = i;
    end
end
% imprimir os valores
name = dic{id_1, 2};
surname = dic{id_1, 3};
fullname = name + " " + surname;
fprintf("User ID: %d; Name: %s\n", id_1, fullname);
name = dic{id_2, 2};
surname = dic{id_2, 3};
fullname = name + " " + surname;
fprintf("User ID: %d; Name: %s\n", id_2, fullname);
```

Opção 4

O utilizador insere uma string e obtém os 3 filmes que possuem os títulos mais similares ao introduzido e devolve o número de vezes que este foi avaliado com nota superior ou igual a 3

No início da implementação desta função, no script **data**, é lido o ficheiro **dic2** que possui os nomes de todos os filmes, algo essencial para este problema. O *array* **film_titles** é então preenchido com os nomes dos filmes em cada um dos seus índices. É declarado um número de *hash_functions*, neste caso 100, e é então chamada a nossa primeira função **calculate_minhash_titles**, que recebe como parâmetro o *array* e a variável referidos.

```
%% option 4
film_titles = [];
for i = 1:length(dic2)
    film_titles = [film_titles string(dic2{i,1})]; % guarda o nome dos filmes
end
n_hash_functions = 100;
minhash_titles = calculate_minhash_titles(film_titles, n_hash_functions);
```

Esta função, é também semelhante às outras duas já utilizadas, no entanto, esta utiliza *shingles* para a obtenção da matriz MinHash. O tamanho destes *shingles*, bem como o de algumas variáveis já utlizadas será explicado ainda neste relatório.

Primeiramente, utilizando o *array* **film_titles** obtido anteriormente, são percorridos todos estes títulos. Cada iteração do ciclo *for* representa um título novo, para o qual serão obtidos através de partes da string, *shingles*, valores de assinaturas para colocar na matriz de MinHash. Utilizando as *hash functions* em cada um dos *shingles* que constituem o título, obtemos então as assinaturas. Estas apenas serão usadas no script **main** uma vez que a *string* introduzida pelo utilizador se encontra lá.

```
function minhash_titles = calculate_minhash_titles(film_titles, n_hash_functions)
    % neste caso e necessario o uso de shingles para os titulos
    size_of_a_shingle = 3;
    minhash_titles = inf(length(film_titles), n_hash_functions);
    h = waitbar(0,'Calculating...');

    for i = 1:length(film_titles)
        waitbar(i/length(film_titles),h);
        film_title = film_titles(i);
        film_title = char(film_title);
        s_length = strlength(film_title) - size_of_a_shingle + 1;

        for j = 1: s_length
            film_shingle = lower(char(film_title(j:(j + size_of_a_shingle - 1))));
            indexes = zeros(1, n_hash_functions);

            for z = 1:n_hash_functions
                film_shingle = [film_shingle num2str(z)];
                indexes(z) = DJB31MA(film_shingle, 181); % numero primo alto
            end
            minhash_titles(i,:) = min([minhash_titles(i,:) ; indexes]);
        end
    end
    delete(h);

    % de seguida é necessario calcular as similaridades
    % no entanto não possuimos a string introduzida para tal
end
```

No script **main**, utilizando a matriz MinHash obtida, é necessário calcular então as assinaturas da *string* introduzida pelo *user* e comparar com a matriz total.

Para obter então a matriz MinHash da *string*, é aplicado exatamente o mesmo conceito, a diferença é que agora é apenas para esta *string*.

```
case 4
    % user pesquisa por um nome de um filme ou parte do nome e
    % obtem os titulos parecidos com este
    n_hash_functions = 100;
    film_name = input("Search for a film: ");
    similar_titles = calculate_similar_titles(minhash_titles, film_name, n_hash_functions, film_titles, ratings);
```

```
function similar_titles = calculate_similar_titles(minhash_titles, film_name, n_hash_functions, film_titles, ratings)
    % temos o minhash titles que tem os hashcodes para cada filme
    % agora temos de achar a distancia entre estes e a string introduzida
    film_name = char(film_name);
    % primeiramente é necessario calcular a matrix minhash da string
    size_of_a_shingle = 3;
    minhash_string = inf(1, n_hash_functions); % desta vez apenas de tamanho 1 pois e so 1
    s_length = strlength(film_name) - size_of_a_shingle + 1;

    for j = 1: s_length
        string_shingle = lower(char(film_name(j:(j + size_of_a_shingle - 1))));
        indexes = zeros(1, n_hash_functions);

        for z = 1:n_hash_functions
            string_shingle = [string_shingle num2str(z)];
            indexes(z) = DJB31MA(string_shingle, 181); % numero primo alto
        end
        minhash_string(1,:) = min([minhash_string(1,:) ; indexes]);
    end
```

De seguida, através de blocos condicionais, obtemos os 3 filmes mais próximos ao introduzido. Percorrendo todas as distâncias obtidas na matriz de similaridade, vamos verificar se uma distância é menor que alguma das três menores distâncias guardadas até então. Caso seja menor que a primeira, a terceira menor passa a ser a segunda menor e esta passa a ser a até então maior distância, uma vez que a primeira obtém uma nova distância menor. Caso seja a segunda menor, apenas é preciso alterar a terceira menor para receber o valor da segunda, e esta recebe um novo valor menor. Se for a terceira menor, apenas atualiza a sua informação.

```
    % agora temos de obter os 3 mais proximo
    sim_title_1 = 0; % id do filme mais proximo
    min_title_1 = 1; % menor distancia
    sim_title_2 = 0;
    min_title_2 = 1;
    sim_title_3 = 0;
    min_title_3 = 1;

    for i = 1:length(similar_titles)
        title_distance = similar_titles(i);
        if title_distance < min_title_1
            % tem de dar o seu valor ao segundo e o segundo ao terceiro
            sim_title_2 = sim_title_1;
            min_title_2 = min_title_1;
            sim_title_3 = sim_title_2;
            min_title_3 = min_title_2;

            sim_title_1 = i;
            min_title_1 = title_distance;

        elseif title_distance < min_title_2
            % igual ao de cima
            sim_title_3 = sim_title_2;
            min_title_3 = min_title_2;

            sim_title_2 = i;
            min_title_2 = title_distance;
        elseif title_distance < min_title_3
            sim_title_3 = i;
            min_title_3 = title_distance;
        end
    end
```

Caso a *string* introduzida não possua nenhum filme mais próximo (todos com distância de 1), o grupo decidiu que os filmes a serem apresentados seriam portanto aleatórios, e todos diferentes.

```
% caso um deles seja 1 (ou todos) damos um filme aleatório
if sim_title_1 == 1
    sim_title_1 = randi([1 1682]);
end

if sim_title_2 == 1
    sim_title_2 = randi([1 1682]);
    while sim_title_2 == sim_title_1 % nao atribuir o mesmo valor
        sim_title_2 = randi([1 1682]);
    end
end

if sim_title_3 == 1
    sim_title_3 = randi([1 1682]);
    while sim_title_3 == sim_title_2 || sim_title_3 == sim_title_1
        sim_title_3 = randi([1 1682]);
    end
```

Esta função continua, pois é necessário obter o numero de *users* que avaliaram o filme com uma nota igual ou superior a 3, no entanto é necessária informação que se encontra no outro script. Esta informação é referente á criação preenchimento de um *Counting Bloom Filter*.

Exisitem três funções necessárias para a realização deste filtro de Bloom, a que inicializa o filtro, a que insere os valores no filtro e a que verifica se um certo valor possui ao filtro.

Primeiramenete, temos a função **initialize**, que apenas tem como objetivo inicializar um *array* com zeros para um tamanho dado.

```
function F = initialize(n)
    F = zeros(1, n);
end
```

Em seguida, temos a função **insert_to_filter** que como dito anteriormente, adiciona um elemento ao filtro. Através de um *for* com o número de funções de dispersão atribuído, este gera um determinado *hash code*, incrementando o filtro nas posições obtidas.

```
function F = insert_to_filter(F,element, n_hash_fuctions)
    for i = 1:n_hash_fuctions
        key = [element num2str(i)];
        index = mod(string2hash(key), length(F))+1;
        F(index) = F(index) + 1;
    end
end
```

A última função, **member**, é bastante parecida com a que insere, no entanto, ao invés de inserir verifica se este item existe e neste caso, sendo um filtro Bloom de contagem, devolve o número de vezes que este item foi colocado no filtro.

Da mesma forma que a anterior são obtidos *hash codes* para cada iteração do número de funções de dispersão. No final obtém o número mínimo que dos valores obtidos, evitando assim grande parte dos falsos positivos.

```
function counter = member(F, element,n_hash_functions)
    filter_counter = zeros(1,n_hash_functions);
    for i = 1:n_hash_functions
        key = [element num2str(i)];
        index = mod(string2hash(key), length(F))+1;
        filter_counter(i) = F(index);
    end
    counter = min(filter_counter); % é usado o valor minimo (minimum selection)
end
```

Utilizando estas funções, o filtro é então iniciado com um tamanho 10000 uma vez que existem 1683 filmes e 6 funções de dispersão. Após a inicialização, os títulos são adicionados, pelas avaliações realizadas no ficheiro **u.data**, ao filtro caso a sua nota seja igual ou superior a 3. Por fim, percorrendo os nomes dos filmes, guarda numa variável **ratings** o resultado obtido pela função **member**, ou seja, o número de vezes que o filme foi avaliado com nota 3 ou superior.

```
% é ainda necessário realizar o Bloom Filter Counter
% inicializa o filtro
filter_size = 10000; % sendo o numero de filmes 1632
n_hash_functions = 6;
filter = initialize(filter_size);
h = waitbar(0,'Filling Bloom Filter...');
for i = 1:length(u)
    waitbar(i/length(u),h);
    % insere o filme se o seu rating for superior ou igual a 3
    if u(i,3) < 3 % caso o rating seja inferior a 3 passa á frente
        continue;
    end
    film_id = u(i,2); % temos o id queremos o nome
    film = film_titles(film_id);
    film = char(film);
    % insere para o filtro de bloom contando
    filter = insert_to_filter(filter, film, n_hash_functions);
end
delete(h)
% obtemos assim o filtro de bloom de contagem
% queremos agora saber o numero de vezes que o filme foi portanto avaliado
ratings = zeros(1, length(film_titles));
h = waitbar(0,'Getting Rates...');
for i = 1:length(film_titles)
    waitbar(i/length(film_titles),h);
    film = film_titles(i);
    film = char(film);
    n_rates = member(filter, film, n_hash_functions);
    ratings(i) = n_rates;
end
delete(h)
```

Finalmente, voltando à função **similar_titles** que se encontra no script **main**, basta imprimir no ecrã os três filmes mais similares à *string* introduzida e o número de notas iguais ou superiores a 3 que este possui.

```
film_1 = film_titles(sim_title_1);
film_2 = film_titles(sim_title_2);
film_3 = film_titles(sim_title_3);
fprintf("Film name: %s; ID: %d; Number of ratings above or equal to 3 (1-5): %d\n", film_1,sim_title_1,ratings(sim_title_1));
fprintf("Film name: %s; ID: %d; Number of ratings above or equal to 3 (1-5): %d\n", film_2,sim_title_2,ratings(sim_title_2));
fprintf("Film name: %s; ID: %d; Number of ratings above or equal to 3 (1-5): %d\n", film_3,sim_title_3,ratings(sim_title_3));
```

Números considerados

Ao longo do relatório foram utilizados vários números que resultaram de escolhas tomadas pelo grupo. O número de funções de dispersão que foram utilizadas no cálculo de todas as matrizes MinHash foi 100. O grupo experimentou com outros valores menores como 50 e maiores como 150 ou até 200, e concluiu que o valor em que os resultados estavam a ser mais visíveis era portanto 100.

No cálculo da matriz MinHash com *shingles* o grupo decidiu usar um comprimento de 3. Foi dada uma dica que deveria ser entre 2 e 5 e após a observação dos títulos, não sendo muito grandes, ficou decidido então um comprimento de 3.

Para obtenção do filtro de Bloom o grupo decidiu usar um tamanho de 10000, uma vez que para o número de filmes que eram, 1683, e utilizando 6 funções de dispersão, seria um bom número. Este número de funções de dispersão foi decidido após se considerar que o tamanho do filtro a ser usado era de 10000.

Ao longo do trabalho foram utilizadas 2 tipos de funções de dispersão, a **DJB31MA** e a **string2hash**. O grupo verificou que a primeira foi a que obteve melhor desempenho durante as aulas, foi utilizada quase sempre, sendo que a segunda apenas foi utilizada dentro do filtro de Bloom tal como também foi no decorrer das aulas práticas.

Resultados obtidos

Opção 1:

```
Insert Film ID (1 to 1682) :1
1 - Users that evaluated current movie
2 - Suggestion of users to evaluate movie
3 - Suggestion of users to based on common interests
4 - Movies feedback based on popularity
5 - Exit
Select choice: 1
Users that rated this movie:
User ID: 308; Name: Adriano Nascimento
User ID: 287; Name: Liane Costa Branco
User ID: 148; Name: Penélope Ferreira
User ID: 280; Name: Ana Pinho
User ID: 66; Name: Yuri Neto
User ID: 5; Name: Iara Amaral
User ID: 109; Name: Marcelo Paiva
User ID: 181; Name: Eunice Neto Faria
User ID: 95; Name: Isis Andrade
User ID: 268; Name: Naiara Carneiro
User ID: 189; Name: Jorge Moura
User ID: 145; Name: Rita Nogueira Baptista
User ID: 158; Name: Arthur Nunes
User ID: 67; Name: Clara Paiva Matias
User ID: 232; Name: Denilson Batista
User ID: 150; Name: Alma Vieira
User ID: 289; Name: Natanael Campos
User ID: 117; Name: Leila Garcia Moura
User ID: 49; Name: Hugo Almeida
User ID: 223; Name: Estêvão Coelho
User ID: 56; Name: Matteo Leal
User ID: 17; Name: Lúcia Nascimento
User ID: 340; Name: Paula Melo Domingues
User ID: 177; Name: Amadeu Paiva
```

Opção 2:

```
Insert Film ID (1 to 1682) :1
1 - Users that evaluated current movie
2 - Suggestion of users to evaluate movie
3 - Suggestion of users to based on common interests
4 - Movies feedback based on popularity
5 - Exit
Select choice: 2
Similar film 1: 50
Similar film 2: 121
Possible users:
User ID: 8; Name: Abel Domingues Maia
User ID: 227; Name: Mia Sousa
User ID: 318; Name: Oriana Alves
User ID: 80; Name: Mafalda Monteiro
User ID: 28; Name: Mateus Fonseca
User ID: 267; Name: Ibrahim Valente Garcia
User ID: 316; Name: Tamára Lima Gaspar
User ID: 48; Name: Filipe Barros
User ID: 154; Name: Benedita Freitas
User ID: 257; Name: Eunice Almeida Garcia
User ID: 269; Name: Eleonor Ramos
User ID: 123; Name: Jorge Leite
User ID: 368; Name: James Leite
User ID: 321; Name: Irina Baptista Simões
User ID: 68; Name: Pilar Guerreiro
User ID: 214; Name: Juliana Carvalho Garcia
User ID: 288; Name: Kelvin Azevedo
User ID: 103; Name: Isabel Marques Nascimento
User ID: 409; Name: Augusto Ribeiro
User ID: 386; Name: Joabe Leite
User ID: 283; Name: Violeta Vicente
```

Opção 3:

```
Insert Film ID (1 to 1682) :1
1 - Users that evaluated current movie
2 - Suggestion of users to evaluate movie
3 - Suggestion of users to based on common interests
4 - Movies feedback based on popularity
5 - Exit
Select choice: 3
User ID: 384; Name: Dayane Morais
User ID: 103; Name: Isabel Marques Nascimento
```

Opção 4:

```
Insert Film ID (1 to 1682) :1
1 - Users that evaluated current movie
2 - Suggestion of users to evaluate movie
3 - Suggestion of users to based on common interests
4 - Movies feedback based on popularity
5 - Exit
Select choice: 4
Search for a film: "man"
Film name: Batman (1989); ID: 403; Number of ratings above or equal to 3 (1-5): 173
Film name: Thin Man, The (1934); ID: 493; Number of ratings above or equal to 3 (1-5): 58
Film name: Third Man, The (1949); ID: 513; Number of ratings above or equal to 3 (1-5): 70
```

Opção 5:

```
Insert Film ID (1 to 1682) :1
1 - Users that evaluated current movie
2 - Suggestion of users to evaluate movie
3 - Suggestion of users to based on common interests
4 - Movies feedback based on popularity
5 - Exit
Select choice: 5
Exixing the program...
>>
```

Conclusão

Na realização deste trabalho prático, consolidamos os nossos conhecimentos sobre esta parte da matéria englobada pelo guião 4, PL4, tanto a nível teórico como a nível prático.

Apesar de algumas dificuldades na realização do mesmo, estas foram ultrapassadas recorrendo ao guião realizado no decorrer das aulas práticas e à material fornecido pelo professor.

Por fim, todos os objetivos deste trabalho foram alcançados, uma vez que todas as opções pedidas foram implementadas com sucesso.