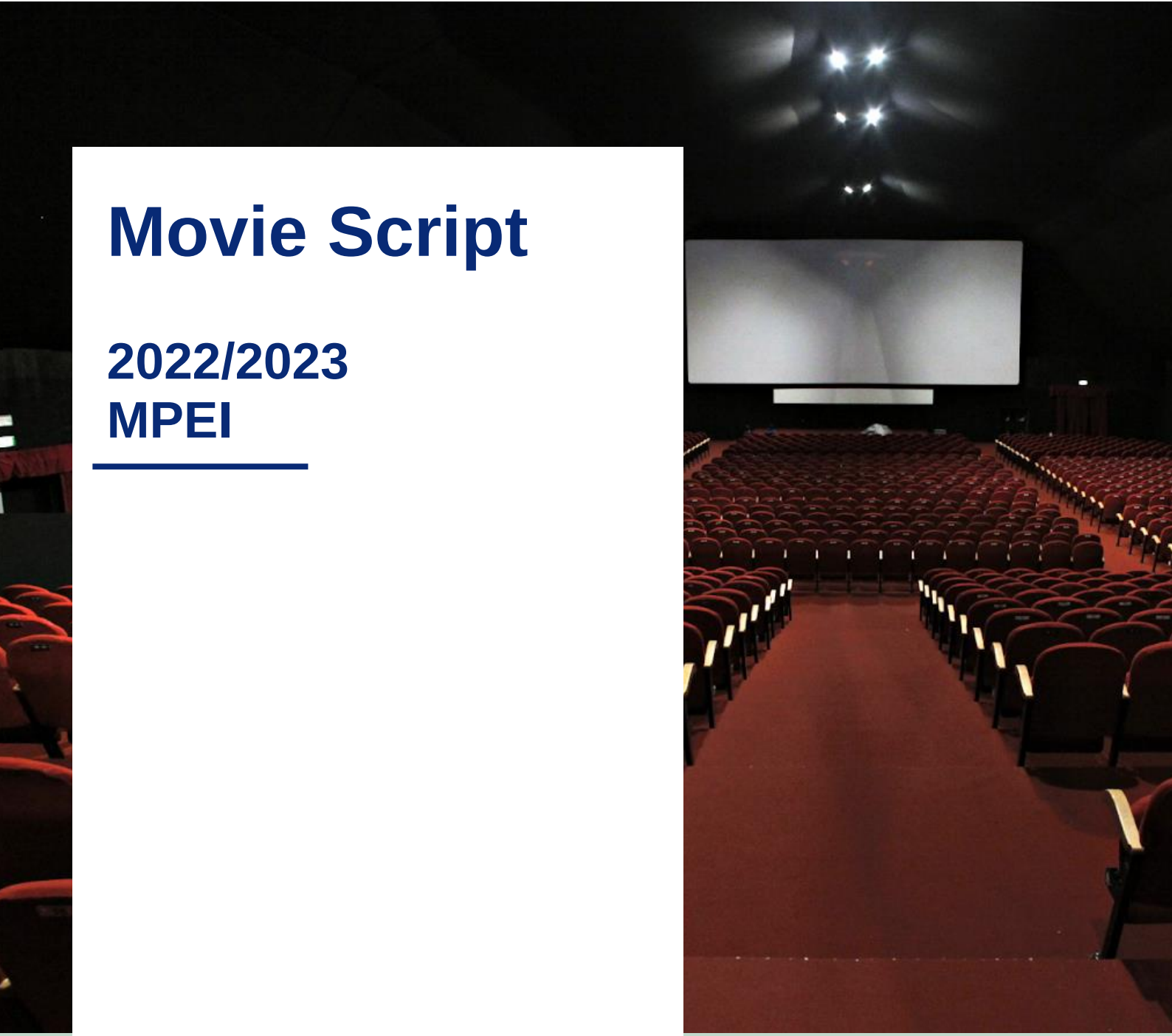


Movie Script

2022/2023
MPEI

2022/2023

Alexandre Cotorobai 107849
Joaquim Rosa 109089

universidade de aveiro

Índice

Introdução

No âmbito da disciplina de Métodos Probabilísticos para Engenharia Informática, foi-nos proposto como trabalho prático o desenvolvimento de uma aplicação, em MATLAB, com algumas funcionalidades de um sistema online de disponibilização de filmes.

Foram nos fornecidos dados sobre os quais tivemos a trabalhar, entre os quais, users.txt com dados sobre cada utilizador, film_info.txt com toda a informação referente aos filmes, e u.data com as avaliações aos filmes dos diferentes utilizadores.

Menu

O ponto de partida desta aplicação é o script read_process.m que vai ler e processar todos os dados e tabelas necessários para que o segundo script (main.m) funcione sem ter que estar a fazer cálculos constantemente.

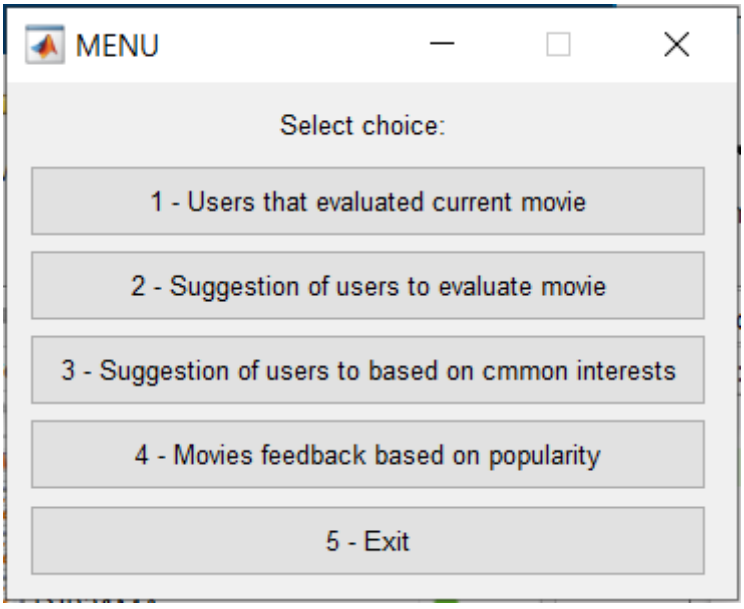
Nesta aplicação será mostrado um menu com 5 opções:

1 - Listar os utilizadores que avaliaram o filme introduzido.

2 - Apresenta os 2 filmes mais similares ao filme introduzido (com maior numero de utilizadores que avaliaram cada filme).

3 - Apresenta os dois utilizadores que com menos 0.9 de distancia de Jacard e que ainda não tenham avaliado o filme atual.

4 - O utilizador insere uma string com o nome (parcial ou completo) e são devolvidos os 3 filmes com o nome mais parecido.



Implementações

Opção 1

- make_Set()

Nesta opção vamos listar os nomes dos utilizadores que avaliaram o filme introduzido.

Inicialmente começamos por criar um set com os dados da "u.data", após ler o ficheiro no read_process.

Nessa função "make_set()" iremos tirar a lista de todos os filmes, sem repetir.

De seguida, percorremos essa lista de filmes e comparamos cada um desses filmes com segunda coluna do u.data (coluna onde estão os filmes que cada utilizador avaliou). No caso de coincidirem (mesmo quando há mais que um), na linha onde se encontrou um filme igual ao da lista iremos pegar no utilizador(es) e adicionar nesse Set.

Deste modo conseguimos ter guardados em cell arrays todos os IDs de utilizadores que viram/avaliaram cada filme.

```
function [Set, Set_Scores, Nu, filmes] = make_set(file)

    % Código base para deteção de pares similares

    udata=file;

    u= udata(1:end,1:3); clear udata;

    filmes = unique(u(:,2));
    Nu= length(filmes);

    Set= cell(Nu,1);
    Set_Scores = cell(Nu,1);
    for n = 1:Nu
        ind = find(u(:,2) == filmes(n));
        Set{n} = [Set{n} u(ind,1)];
        Set_Scores{n} = [Set_Scores{n} u(ind,3)];
    end
end
```

- findUsers()

Já com os dados organizados agora é só procurar no set pelo ID do filme introduzido e imprimir os vários utilizadores associados ao mesmo.

```
function Set = findUsers(Set, id_filme, user_names)
%FINDUSERS Summary of this function goes here
%   Detailed explanation goes here
    users = Set{id_filme};
    for i = 1:length(users)
        user_id = users(i);
        fprintf("%d - %s\n", user_id, user_names{user_id});
    end
end
```

Opção 2

Na opção 2 é pedido para listar os 2 filmes mais similares ao introduzido, com base nos users que viram o filme introduzido.

```
        findUsers(Set, id, users_names);

    case 2
        disp("Executing 2")

        [J] = distancia(MinHash2, Set, id, filmes_info, users_names);

    case 3
        disp("Executing 3")
```

- minHashUsers()

A função de hash usada para fazer a minHashUsers, foi uma que estava disponível no livro recomendado: "Métodos Probabilísticos para Engenharia Informática", apenas com umas pequenas adaptações.

```
function MinhHashUsers = minHashUsers(Set,nfd)

v  = inicFuncoesDispersao;

nc = length(Set);

MinhHashUsers = zeros(nfd, nc);

for nu = 1:nc
    C = Set{nu};

    for nh = 1:nfd
        MinhHashUsers(nh,nu) = mod(v.a(nh)*C(1)+v.b(nh), v.p);

        for nf = 2:length(C)
            htmp = mod(v.a(nh)*(C(nf))+v.b(nh), v.p);

            if htmp < MinhHashUsers(nh,nu)
                MinhHashUsers(nh,nu) = htmp;
            end
        end
    end
end
```

- distancia()

Essa função calcula a distância entre dois filmes usando a MinHash. A distância entre os filmes é medida pela distância de Jaccard.

Aqui recebemos como argumentos a MinHash, o Set com os utilizadores que viram cada filme, id_filme inserido, filmes_info com todos os dados dos filmes e o user_names.

A função inicializa um array J com tamanho igual ao número de filmes. Em seguida, usamos um loop para calcular a distancia de Jaccard entre cada filme e o filme atual.

A distância de Jaccard é calculada com recurso a logical indexing, fazendo a soma dos valores lógicos resultantes da comparação dos valores da minHash e dividindo pelo tamanho de uma coluna da mesma. Depois, a função ordena os filmes de acordo com sua distância com o filme atual e exibe os dois filmes mais similares.

Além disso, ela exibe os usuários que avaliaram os filmes similares, mas não avaliaram o filme atual (id_filme).

```
function [J] = distancia(MinHash, Set, id_filme, filmes_info, user_names)
    Nu = length(Set);
    J=zeros(1,Nu);   % array para guardar distancias
    h= waitbar(0,'Calculating');
    for n1= 1:Nu
        waitbar(n1/Nu,h);
        J(n1) = 1 - (sum(MinHash(:,n1)==MinHash(:,id_filme))/length(MinHash(:,1)));
    end
    delete (h)

    [J,index] = sort(J);

    disp("Filmes Similares:")
    fprintf("ID %d - %s\nID %d - %s\n", index(2), filmes_info{index(2),1}, index(3), filmes_info{index(3),1});

    users_of_similares = union(Set{index(2)}, Set{index(3)});
    diff_users = setdiff(users_of_similares, Set{id_filme});

    if ~isempty(diff_users)
        disp("Users que ainda não avaliaram o filme:")
        for i = 1:length(diff_users)
            fprintf("ID %d - %s\n", diff_users(i), user_names{diff_users(i)});
        end
    else
        disp("Todos os users já avaliaram o filme atual")
    end
end
```

Opção 3

Aqui vamos filtrar os gêneros dos vários utilizadores e passá-los para um set, removendo qualquer valor nulo presente (pois o número de gêneros por utilizador varia).

```
%% Opção 3
Set_generos = make_set_generos(users);
for i = 1:length(Set_generos)
    set =Set_generos{i,1};
    x = cellfun(@numel,set);
    set(x==1) = [];
    Set_generos{i,1}= set;
end
distancia_user_generos = distanciaGeneros(Set_generos, nhf);
%% Opcao 4
```

No script main.m é onde iremos ordenar ao utilizadores que não viram/avaliaram o filme introduzido.

```
case 3
    disp("--------------- OPÇÃO 3 ---------------")
    nao_viram_o_filme = simUsersFinder(id, distancia_user_generos, udata, Set);

    [sorted, index] = sort(values(nao_viram_o_filme), "descend");
    usersId = keys(nao_viram_o_filme);
    fprintf("%d - %s\n", usersId(index(1)), users_names{usersId(index(1))});
    fprintf("%d - %s\n", usersId(index(2)), users_names{usersId(index(2))});
```

-make_set_generos()

Essa função cria um cell array de conjuntos de gêneros de filmes para cada usuário.

A função primeiro extrai as colunas com os gêneros contidos em users. Em seguida, cria um cell array vazio chamado Set_generos, para poder guardar os generos de cada um dos utilizadores. Depois, usa um loop para preencher cada elemento de Set_generos com o conjunto de gêneros avaliados pelo user correspondente.

No final obtemos o Set_generos, que é um cell array em que cada elemento é o conjunto de gêneros avaliados pelo user.

```
function Set_generos = make_set_generos(file)

    users=file;

    u= users(1:end,4:end); clear users;

    Nu= length(u);

    Set_generos= cell(Nu,1);
    for id = 1:Nu
        Set_generos{id} = [Set_generos{id} u(id,:)];
    end
end
```

-distancia_user_generos()

Essa função calcula a distância entre os conjuntos de gêneros de filmes avaliados por cada par de usuários.

A função inicializa uma matriz MinHashGeneros. Em seguida, usa um loop para calcular o MinHash de cada usuário. Para isso, ela usa uma série de funções de hash e o conjunto de gêneros avaliados pelo usuário para calcular o MinHash.

```
function distancia_user_generos = distanciaGeneros(Set_generos, nhf)
    Nu = length(Set_generos);
    seedMatrix = randi([1 1000],1,nhf);
    MinHashGeneros = zeros(Nu,nhf);
    v  = inicFuncoesDispersao;

    h = waitbar(0,'A calcular minHashGeneros()...');
    for user=1:Nu
        waitbar(user/Nu,h);
        for hf=1:nhf
            Generos = Set_generos{user,1};
            hashArr=zeros(1,length(Generos));

            for genero_index = 1:length(Generos)
                key = mod(v.a(hf)*sum(double(Generos{genero_index}))+v.b(hf), v.p);
                hashArr(genero_index) = rem(DJB31MA(key, seedMatrix(hf)), 7919)+1;
            end
            MinHashGeneros(user,hf) = min(hashArr);
        end
    end
    delete(h);

    distancia_user_generos = zeros(Nu,Nu); % array para guardar distancias

    for n1= 1:Nu
        for n2= 1:Nu
            distancia_user_generos(n1,n2) = 1-sum(MinHashGeneros(n1,:)==MinHashGeneros(n2,:))/length(MinHashGeneros(n2,:));
        end
    end
end
```

- inicFuncoesDispersao()

Esta função inicializa as funções de dispersão para serem usadas no cálculo da MinHash dos gêneros dos utilizadores. A função define dois parâmetros, m e nfd, que são usados para determinar o tamanho da tabela hash e o número de funções de dispersão que serão geradas, respetivamente.

```
function v = inicFuncoesDispersao
    m = 100000;
    nfd = 200;

    ff = 1000;
    pp = ff * max(m + 1,76);
    pp = pp + ~mod(pp,2);
    while(isprime(pp) == false)
        pp = pp + 2;
    end
    v.p = pp;

    v.a = randi([1, (pp-1)],1,nfd);
    v.b = randi([0, (pp-1)],1,nfd);
    v.c = randi([1, (pp-1)],1,nfd);
end
```

Depois, a função inicializa uma matriz distancia_user_generos para armazenar a distância entre os conjuntos de gêneros de cada par de usuários. Em seguida, usa outro loop para calcular a distância entre os conjuntos de gêneros de cada par de usuários. A distância é calculada como 1 menos a proporção de hashes iguais entre os dois usuários.

Portanto, a função retorna uma matriz distancia_user_generos, em que cada elemento (n1,n2) representa a distância entre os conjuntos de gêneros avaliados pelos usuários n1 e n2.

- simUsersFinder()

```
function nao_viram_o_filme = simUsersFinder(id_filme,distancia_user_generos, udata, Set)
%BERNARDO Summary of this function goes here
%   Detailed explanation goes here

    ind = find(udata(:,2) == id_filme);

    users = udata(ind(:,1)); %users que viram o filme
    nao_viram_o_filme = dictionary([],[]);
    for i=1:length(users)
        userDist = distancia_user_generos(users(i),:); %para cada user ve as distancias
        indOfDist = userDist < 0.9;
        indOfDist(users(i)) = 0; % remove user himself from index;
        simUsers = find(indOfDist);

        for j=1:length(simUsers)
            watchedFilm = ismember(Set{id_filme},simUsers(j));
            if watchedFilm == zeros(length(Set{id_filme}),1)    %se não viu o filme
                if isKey(nao_viram_o_filme , simUsers(j))
                    nao_viram_o_filme(simUsers(j)) = nao_viram_o_filme(simUsers(j))+1;
                else
                    nao_viram_o_filme(simUsers(j)) = 1;
                end
            end
        end
    end

end
```

Esta função é usada para encontrar users que não viram um determinado filme, mas que são similares a outros users que viram o filme. A função recebe como entrada o ID do filme, uma matriz de distância entre users e gêneros, uma matriz de dados de users e um conjunto de filmes vistos por cada usuário.

A função começa encontrando os índices na matriz de dados de users correspondentes ao filme especificado. Em seguida, ela itera sobre cada users que viu o filme, encontra os users similares a esse usuário (com base em uma distância menor que 0,9) e verifica se esses users similares também viram o filme. Se eles não viram, são guardados como keys num dicionário, tendo como value o número de vezes que eles vão aparecer no conjunto de utilizadores similares dos utilizadores que viram o filme. O nome e o ID dos dois utilizadores que aparecerem mais vezes nos conjuntos serão imprimidos no terminal.

```
[sorted, index] = sort(values(nao_viram_o_filme), "descend");
usersId = keys(nao_viram_o_filme);
fprintf("%d - %s\n", usersId(index(1)), users_names{usersId(index(1))});
fprintf("%d - %s\n", usersId(index(2)), users_names{usersId(index(2))});
```

Opção 4

- MinHashFilmNames()

```
function MinHashFilmNames = minHashFilmNames(filmes_info, nhf, shingle_Size, RandomSeeds)
    Nf = length(filmes_info);
    MinHashFilmNames = inf(Nf,nhf);

    h = waitbar(0,'A calcular minHashFilmNames()...');
    for f = 1:Nf
        waitbar(f/Nf,h);
        film = string(filmes_info{f,1});

        for hf = 1:nhf
            x = zeros(1,(strlength(film) - shingle_Size + 1)); % nr de shingles

            for j = 1:(strlength(film) - shingle_Size + 1)
                shingle = lower(char(extractBetween(film,j,j+shingle_Size-1)));

                hc = rem(DJB31MA(shingle,RandomSeeds(hf)),7919)+1;

                x(j) = hc;
            end
            MinHashFilmNames(f,hf) = min(x);
        end
    end
    delete(h)
end
```

Esta função é usada para calcular o valor MinHash de um conjunto de nomes de filmes. O valor MinHash é um valor numérico que é uma aproximação da similaridade entre dois nomes de filmes. A função recebe como entrada uma matriz de informações sobre os filmes, o número de valores MinHash que devem ser calculados para cada filme, o tamanho dos shingles (que são sequências de caracteres) que serão usados para calcular o valor MinHash e uma seed aleatória para ser usada no cálculo.

- CountBloomFilter()

```
function BloomMatrix = CountBloomFilter(BloomMatrix, bloomfilterHashMatrix, Set_Scores,id,n,nhf)
    F = zeros(n,1);

    for score = 1:length(Set_Scores)
        for hf = 1:nhf
            hashcode = rem(DJB31MA(Set_Scores(score),bloomfilterHashMatrix(hf)),n)+1;
            F(hashcode) = F(hashcode)+1;
        end
    end

    BloomMatrix(:,id) = F;
end
```

Esta função cria uma matriz de dimensões (número de funções de hash x número de filmes), sendo que cada coluna corresponde ao filtro de Bloom de cada filme, e os elementos que são colocados no filtro é as classificações de 1 a 5 que o filme recebeu. Cada vez que é adicionada uma classificação a um filtro de Bloom, este incrementa 1 nos índices do filtro correspondentes a essa classificação, de modo fazer umas contagens das classificações que cada filme recebeu.

- findFilm()

```
function findFilm(film_name, MinHashFilmNames, shingle_Size, nhf, filmes_info, RandomSeeds, BloomMatrix, bloomfilterHashMatrix, k)
    hashcodes = zeros(1,(strlength(film_name) - shingle_Size + 1));
    load("u.data")
    film_name_minhash = zeros(1,nhf);

    for hf = 1:nhf

        for j = 1:(strlength(film_name) - shingle_Size + 1)
            shingle = lower(char(extractBetween(film_name,j,j+shingle_Size-1)));

            hc = rem(DJB31MA(shingle,RandomSeeds(hf)),7919)+1;

            hashcodes(j) = hc;
        end
        film_name_minhash(hf) = min(hashcodes);
    end

    distance_film_name = zeros(1,length(filmes_info));

    for f= 1:length(filmes_info)
        distance_film_name(f) = 1 - (sum(MinHashFilmNames(f,:)==film_name_minhash(1,:))/length(MinHashFilmNames(1,:)));
    end


    [distance_film_name,index] = sort(distance_film_name);
    flip(distance_film_name);
    flip(index);

    for i = 1:3
        total = 0;
        for elem = 3:5
            Avaliacoes = numAvaliacoes(BloomMatrix, elem, k, bloomfilterHashMatrix, index(i));
            total = total + Avaliacoes;
        end

        fprintf("%s - Número de vezes avaliado com nota igual ou superior a 3: %d\n",filmes_info{index(i),1},total)
    end
end
```

Esta função é usada para encontrar filmes similares a um filme de entrada dado o seu nome. A função recebe como entrada o nome do filme, uma matriz de valores MinHash para todos os filmes, o tamanho dos shingles usados para calcular os valores MinHash, o número de hash functions que serão aplicadas aos shingles, informações sobre todos os filmes, uma seed aleatória para ser usada no cálculo dos valores MinHash, a matriz com filtros de Bloom para cada filme e o array de hashes para a matriz de filtros de Bloom.

A função começa calculando os valores MinHash para o filme de entrada da mesma maneira que a função minHashFilmNames. Em seguida, ela calcula a distância entre o filme de entrada e cada outro filme usando os valores MinHash. A distância é calculada como 1 - (o número de valores MinHash iguais entre os dois filmes / o número total de valores MinHash). A função então classifica os filmes de acordo com a distância e imprime os três filmes mais similares e o número de vezes que cada filme foi avaliado com uma nota igual ou superior a 3, com a utilização da BloomMatrix e da bloomfilterHashMatrix.

Escolha de números

Em relação ao número de funções de dispersão chegamos a conclusão, após vários testes, que 200 seria o número mais sensato a usar visto que acima desse valor não haviam melhoras significativas na precisão da MinHash e como a velocidade de processamento não foi muito afetada não sentimos necessidade de diminuir esse número, ficando assim definido a 200.

Quanto ao tamanho dos shingles recorremos a testes onde, após várias tentativas, chegamos a conclusão que para shingles com tamanho 3 obtemos os melhores resultados.

A fórmula usada para o número de funções de dispersão ideal foi a seguinte:

nhf = (Tamanho do filtro / Número de elementos a colocar no filtro) * log(2)

Conclusão

Com a realização deste guião conseguimos colocar em prática tudo que nos foi ensinado nas aulas práticas, aprofundamos conhecimentos sobre a matéria e ficamos mais proficientes com os algoritmos em causa.

Para finalizar, podemos dizer que conseguimos alcançar grande parte dos objetivos propostos e consideramos ter tido um resultado satisfatório visto o tempo dado para a execução do mesmo.

Bibliografia

- https://www.mathworks.com/
- TEIXEIRA, António; VAZ, Francisco. Métodos Probabilísticos para a Engenharia Informática. 1 ed. Lisboa: Edições Sílabo, 2021.