MPEI PL4 - Relatório

Universidade de Aveiro(*UA*)

Anderson Lourenço, Bernardo Marujo

PL4

Métodos Probabilísticos para Engenharia Informática

DETI - Departamento de Electrónica, Telecomunicações e Informática

Anderson Lourenço, Bernardo Marujo

(108579) aaokilourenco@ua.pt,

(107322)bernardomarujo@ua.pt

21/12/2023

Índice

Índice **1**

1 Introdução **2**

2 Menu **3**

3 Opções Implementadas **4**

4 Conclusão **15**

Capítulo 1

Introdução

O objetivo dste trabalho prático sobre algoritmos probabilísticos é o desenvolvimento de uma app em Matlab (sistema de disponibilização de filmes).

As tarefas que estarão no menu do programa serão as seguintes:

1. Listar os géneros de filmes disponíveis
2. Número de filmes por género
3. Número de filmes por género num dado ano
4. Procurar filmes por título
5. Procurar filmes por géneros

Capítulo 2

Menu

Para este Menu, foi usado, detro de um while true loop, com a função input() do MATLAB seguida de um switch case, para obter o input do utilizador.

O seguinte código (presente no script2.m) demonstra um pseudocódigo de como foi feito:

```
    while(1)
        option = input(['\n1 - Display available genres' ...
                        '\n2 - Number of movies of a genre' ...
                        '\n3 - Number of movies of a genre on a
  given year' ...
                        '\n4 - Search movie titles' ...
                        '\n5 - Search movies based on genres' ...
                        '\n6 - Exit' ...
                        '\nSelect an option: ']);

        switch option
```

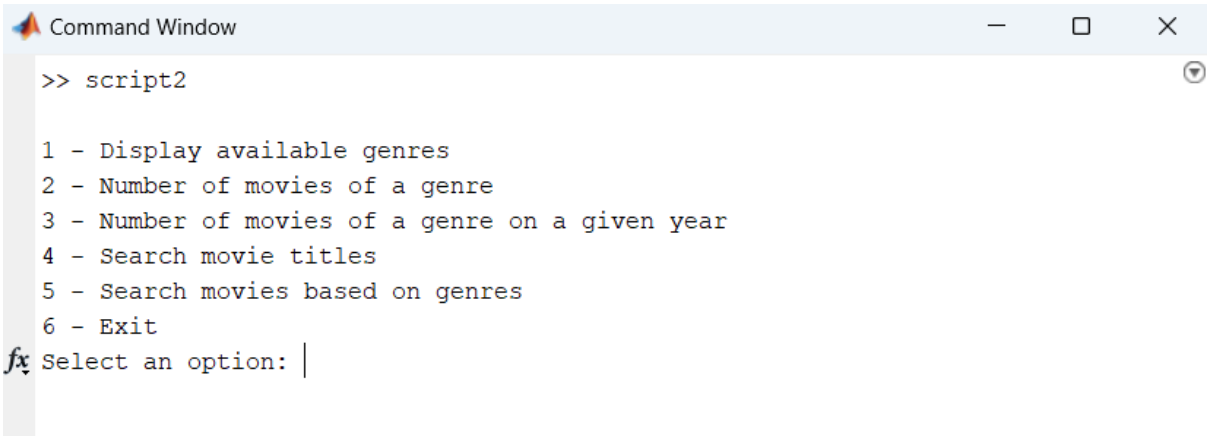

Figura 2.1: Menu da aplicação

Capítulo 3

Opções Implementadas

Option 1

A função chamada getGenres recebe uma tabela movies como entrada e retorna uma matriz de células de gêneros contendo gêneros únicos.

A função começa inicializando uma matriz de células vazia, gêneros e um contador k definido como 1. A matriz de células gêneros será usada para armazenar os gêneros encontrados na tabela de filmes e k é usado para rastrear o índice atual na matriz de gêneros.

Em seguida, a função entra em um loop aninhado que itera sobre cada linha (ou seja, cada filme) e colunas específicas (da 3ª à 7ª) da tabela de filmes. Presume-se que essas colunas contenham as informações de gênero de cada filme.

Dentro do loop interno, a função verifica se a célula atual da tabela não está faltando (usando a função anymissing) e não é igual à string 'unknown'. Se ambas as condições forem verdadeiras, o gênero é adicionado à matriz de células de gêneros no índice atual k e então k é incrementado em 1.

Finalmente, após todos os filmes e seus respectivos gêneros terem sido processados, a função usa a função exclusiva para remover quaisquer gêneros duplicados da matriz de células de gêneros. Essa matriz final desduplicada é então retornada como saída da função getGenres.

```
genres = getGenres(movies);

function genres = getGenres(movies)
    genres = {};
    k = 1;

    for i = 1:height(movies)
        for j = 3:7
            if ~anymissing(movies{i, j}) && ~strcmp(movies{i,
j}, 'unkown')
                genres{k} = movies{i, j};
                k = k + 1;
            end
        end
    end
```

```
        genres = unique(genres);
    end
```

O código que segue é para o menu presente no script2.m:

```
        case 1
            fprintf("Available genres:\n")
            for i = 2:length(genres)
                fprintf("-%s \n", genres{i});
            end
```

Option 2 & 3

Esta opções usam Bloom Filter e o código define quatro funções: init, insert, insert2 e DJB31MA. Essas funções são usadas para criar e manipular um filtro Bloom, que é uma estrutura de dados usada para testar se um elemento é membro de um conjunto.

A função init recebe um número inteiro n como entrada e retorna um filtro Bloom BF inicializado como uma matriz de zeros 1 por n.

A função insert é usada para adicionar um elemento ao filtro Bloom. São necessárias três entradas: elemento (o elemento a ser adicionado), BF (o filtro Bloom) e k (o número de funções hash a serem usadas). O valor ótimo para k foi calculado utilizando a expressão,

$$K_{\text{ótimo}} \approx \frac{n \ln(2)}{m} = \frac{0.693n}{m}$$

tendo nós obtido assim um valor de $k \approx 6$.

A função primeiro calcula o comprimento do filtro Bloom. Então, ele entra em um loop que é executado k vezes. Se elemento não estiver faltando, ele anexa a representação de string do índice do loop atual ao elemento, calcula um valor hash h usando a função DJB31MA e incrementa o h-ésimo elemento do filtro Bloom.

A função insert2 é semelhante a insert, mas também requer uma entrada adicional ano (ano). A principal diferença é que insert2 precede a representação de string de ano ao elemento antes de anexar a representação de string do índice do loop atual.

A função DJB31MA é uma função hash. Ele recebe uma string chave e uma seed como entradas e retorna um valor hash. A função primeiro calcula o comprimento da chave e converte a chave em um array duplo. Em seguida, ele inicializa o valor hash como semente. Finalmente, ele entra em um loop que percorre todo o comprimento da chave, atualizando o valor do hash a cada iteração com base no caractere atual da chave. O valor hash é calculado usando a fórmula,

$$mod(31 * h + chave(i), 2^{32} - 1)$$

que é uma fórmula comum usada em funções hash por sua boa distribuição e propriedades de resistência a colisões.

```
    BF = init(length(genres)*8);

    BF_years = init(length(genres)*length(years)*8);
```

```
 for i = 1:height(movies)
     for j=3:12
         BF = insert(movies{i, j}, BF, k);
         BF_years = insert2(movies{i,j},BF_years,k,movies{i
,2});
     end
 end

 function BF = init(n)
     BF = zeros(1,n);
 end

 function BF = insert(elemento, BF, k)
     n = length(BF);
         for i = 1:k
             if ~ismissing(elemento)
                 elemento = [elemento num2str(i)];
                 h = DJB31MA(elemento, 127);
                 h = mod(h,n) + 1; %para dar valor entre 1 e n
 para por no BF
                 BF(h) = BF(h)+1;
             end
         end
 end

 function BF = insert2(elemento, BF, k, ano)
     n = length(BF);
         for i = 1:k
             if ~ismissing(elemento)
                 elemento = [num2str(ano) elemento num2str(i)
];
                 h = DJB31MA(elemento, 127);
                 h = mod(h,n) + 1; %para dar valor entre 1 e n
 para por no BF
                 BF(h) = BF(h)+1;
             end
         end
 end

 function h= DJB31MA( chave, seed)
     len= length(chave);
     chave= double(chave);
     h= seed;
     for i=1:len
         h = mod(31 * h + chave(i), 2^32 -1) ;
     end
 end
```

A validação do número de géneros é feita através das seguintes funções:

```
function check = valid(elemento, BF, k)
    n = length(BF);
    for i = 1:k
        elemento = [elemento num2str(i)];
        h = DJB31MA(elemento, 127);
        h = mod(h,n) + 1;
        if BF(h)
            check(i) = BF(h);
        else
            check(i) = 0;
        end
    end
end

function check = valid2(elemento, ano, BF, k)
    n = length(BF);
    for i = 1:k
        elemento = [num2str(ano) elemento num2str(i)];
        h = DJB31MA(elemento, 127);
        h = mod(h,n) + 1;
        if BF(h)
            check(i) = BF(h);
        else
            check(i) = 0;
        end
    end
end
```

O código que segue é para o menu presente no script2.m:

```
    case 2
        genre = input("Select a genre: ","s");
        a=0;
        for x = 1:length(genres)
            if strcmp(genre, genres{x})
                a=1;
                break;
            end
        end
        if a==1
            check=min(valid(genre,BF,6));
            fprintf("\nMovies of '%s' genre: %d\n",genre,
check)
        else
            fprintf("\nGenre doesn't exist. Press 1 to see
available genres.\n")
        end

    case 3
```

```
        genre_year = input("Select a genre and a year (
separated by ','): ","s");
        values = strsplit(genre_year, ',');
        a=0;
        b=0;
        for x = 1:length(genres)
            if strcmp(values{1}, genres{x})
                a=1;
                break;
            end
        end
        for x = 1:length(years)
            if str2num(values{2}) == years(x)
                b=1;
                break;
            end
        end

        if a==1 && b==1

            check=min(valid2(values{1},values{2},BF_years,6))
;
            fprintf("\nMovies of genre %s on year %d: %d\n",
values{1},str2num(values{2}),check)

        else
            fprintf("Invalid Inputs\n")
        end
```

Option 4

A função chamada minHashTitles que executa MinHashing em um conjunto de títulos de filmes. MinHashing é uma técnica usada em mineração de dados para estimar a similaridade entre conjuntos.

A função recebe três entradas: títulos (uma matriz de células de títulos de filmes), numHash (o número de funções hash a serem usadas) e shingleSize (o tamanho das telhas a serem usadas). Shingling é uma forma de converter documentos em conjuntos, tratando cada documento como um conjunto de substrings de um determinado comprimento.

A função começa inicializando uma matriz matrizMinHashTitles de tamanho numTitles por numHash com todos os elementos definidos como infinito. Esta matriz armazenará as assinaturas MinHash dos títulos dos filmes.

A seguir, a função cria uma barra de espera x para mostrar o andamento do processo MinHashing. Em seguida, ele entra em um loop que itera sobre cada título de filme.

Dentro do loop externo, a função atualiza a barra de espera e extrai o título do filme atual. Em seguida, ele entra em outro loop que itera sobre cada telha do título do filme. Uma telha é uma substring do título do filme com comprimento shingleSize.

Dentro do loop interno, a função primeiro converte a telha em minúscula e então inicializa um vetor h de zeros de comprimento numHash. Em seguida, ele entra em um terceiro loop que itera numHash vezes. Neste loop, a função anexa a representação de string do índice do loop atual ao shingle, calcula um valor hash usando a função DJB31MA e armazena o valor hash em h.

Após todos os valores de hash terem sido calculados para o shingle atual, a função atualiza a assinatura MinHash do título do filme atual em matrizMinHashTitles tomando o mínimo elemento a elemento da assinatura MinHash atual e o vetor h.

Por fim, após o processamento de todos os títulos de filmes, a função exclui a barra de espera e termina. A matrizMinHashTitles, que agora contém as assinaturas MinHash de todos os títulos de filmes, é a saída da função.

```
    titles = movies(:,1);
    numTitles = length(titles);
    numHash = 100;
    shingleSize = 3;
    matrizMinHashTitles = minHashTitles(titles,numHash,
  shingleSize);

    function matrizMinHashTitles = minHashTitles(titles,numHash,
  shingleSize)
        numTitles = length(titles);
        matrizMinHashTitles = inf(numTitles, numHash);
        x = waitbar(0,'MinHash Titles');
        for k= 1 : numTitles
            waitbar(k/numTitles,x);
            movie = titles{k};
            for j = 1 : (length(movie) - shingleSize + 1)
```

```
                shingle = lower(char(movie(j:(j+shingleSize-1))))
;
                h = zeros(1, numHash);
                for i = 1 : numHash
                    shingle = [shingle num2str(i)];
                    h(i) = DJB31MA(shingle, 127);
                end
            matrizMinHashTitles(k, :) = min([matrizMinHashTitles(
k, :); h]);
            end
        end
        delete(x);
    end
```

O cálculo dos títulos mais similares a uma determinada string é feito através da distância e similaridade de Jaccard usando as assinaturas MinHash. As seguintes funções implementam esta funcionalidade:

```
    function searchTitle(search, matrizMinHashTitles, numHash,
titles, shingleSize,movies)
        minHashSearch = inf(1, numHash);
        for j = 1 : (length(search) - shingleSize + 1)
            shingle = char(search(j:(j+shingleSize-1)));
            h = zeros(1, numHash);
            for i = 1 : numHash
                shingle = [shingle num2str(i)];
                h(i) = DJB31MA(shingle, 127);
            end
            minHashSearch(1, :) = min([minHashSearch(1, :); h]);
        end
        threshold = 0.99;
        [similarTitles,distancesTitles,k] = filterSimilar(
threshold,titles,matrizMinHashTitles,minHashSearch,numHash);

        if (k == 0)
            disp('No results found');
        elseif (k > 5)
            k = 5;
        end

        distances = cell2mat(distancesTitles);
        [distances, index] = sort(distances);

        for h = 1 : k
            fprintf('\n%s - Similarity: %.3f\n', similarTitles{
index(h)}, 1-distances(h));
            index2 = movie_index(titles,similarTitles{index(h)});
            genres_of_movie=movie_genres(index2,movies);
            fprintf("Genres: ");
            for p = 1:length(genres_of_movie)
```

```
                fprintf("%s  ",genres_of_movie{p});
            end
        end
    end

    function [similarTitles,distancesTitles,k] = filterSimilar(
threshold,titles,matrizMinHashTitles,minHash_search,numHash)
        similarTitles = {};
        distancesTitles = {};
        numTitles = length(titles);
        k=0;
        for n = 1 : numTitles
            distancia = 1 - (sum(minHash_search(1, :) ==
matrizMinHashTitles(n,:)) / numHash);
            if (distancia < threshold)
                k = k+1;
                similarTitles{k} = titles{n};
                distancesTitles{k} = distancia;
            end
        end
    end
```

O código que segue é para o menu presente no script2.m:

```
        case 4
            search = lower(input("Insert a string: ","s"));
            fprintf('\n');

            while (length(search) < shingleSize)
                fprintf("String must have at least %d characters\
n", shingleSize);
                search = lower(input("Insert a string: ","s"));
            end

            searchTitle(search, matrizMinHashTitles, numHash,
titles, shingleSize, genres)
```

Option 5

A função matrizAss cria uma matriz de associação entre filmes e gêneros. Toma como entrada uma tabela de filmes e uma lista de gêneros. A função inicializa uma matriz vazia de zeros com dimensões correspondentes ao número de gêneros e ao número de filmes. Em seguida, ele itera cada gênero e cada filme, verificando se o gênero está associado ao filme. Se o gênero estiver associado ao filme, ele define a célula correspondente na matriz como 1.

A função minHash aplica o algoritmo MinHash à matriz de associação criada pelo matrizAss. Toma como entrada a matriz de associação e o número de funções hash a serem utilizadas. A função inicializa uma matriz vazia de zeros com dimensões correspondentes ao número de funções hash e ao número de filmes. Em seguida, aplica o algoritmo MinHash a cada coluna da matriz de associação, armazenando os resultados na matriz MinHash.

A função getDistancesMinHashGenres calcula a similaridade de Jaccard entre cada par de filmes com base em suas assinaturas MinHash. Toma como entrada o número de filmes, a matriz MinHash e o número de funções hash utilizadas. A função inicializa uma matriz vazia de zeros com dimensões correspondentes ao número de filmes. Em seguida, ele itera sobre cada par de filmes, calculando a similaridade de Jaccard entre suas assinaturas MinHash e armazenando o resultado na matriz de distâncias.

Esta opção demora a rodar pois o cálculo das distâncias estão a ser feitas no script2.m, pois é uma matriz 58kx58k.

NOTA: Para testar esta opção mais rapidamente, pode rodar o script1.m com menos linhas do ficheiro movies.csv, para alterar o data.mat e o cálculo das distâncias ser menor.

Exemplo no script1.m:

```
clear;
clc;

movies = readcell('movies.csv', 'Delimiter', ',');
movies = movies(1:2000, :)
k=6;

```

Código para opção 5:

```
numFilms = height(movies);
numHash = 100;
% genres = getGenres(movies);
numGenres = length(genres);
matrizAssGenres = matrizAss(movies,genres);
matrizMinHashGenres = minHash(matrizAssGenres,numHash);
distancesGenres = getDistancesMinHashGenres(numFilms,
matrizMinHashGenres,numHash);

function matrizAss = matrizAss(movies,genres)
    numFilms = height(movies);
    numGenres = length(genres);
    matrizAss = zeros(numGenres,height(movies));

    for i= 1:numGenres
        for n= 1:numFilms
            for k= 2:7
                if ~anymissing(movies{n,k})
                    if strcmp(genres(i),movies{n,k})
                        matrizAss(i,n) = 1;
                    end
                end
            end
        end
    end
end

function matrizMinHashGenres = minHash(matrizAss,numHashFunc)
```

```
      p = primes(10000);
      matrizMinHashGenres = zeros(numHashFunc,width(matrizAss))
;
      kList = p(randperm(length(p),numHashFunc));

      for func= 1:length(kList)
          for d= 1:width(matrizAss)
              matrizMinHashGenres(func,d) = min(mod(find(
matrizAss(:,d)==1),kList(func)));
          end
      end
 end

 %==================script2.m=====================

 function similarGenres = filterSimilarGenres(
user_genre_vector, distancesGenres, threshold)
      numGenres = length(user_genre_vector);
      similarGenres = zeros(numGenres, 3);
      k = 0;
      for n = 1 : numGenres
          if user_genre_vector(n)
              for m = 1 : numGenres
                  if ~user_genre_vector(m)
                      k = k + 1;
                      similarGenres(k, 1) = m;
                      similarGenres(k, 2) = n;
                      similarGenres(k, 3) = distancesGenres(n,
m);
                  end
              end
          end
      end
      similarGenres = similarGenres(1:k, :);
      similarGenres = sortrows(similarGenres, 3, 'descend');
      similarGenres = similarGenres(similarGenres(:, 3) >=
threshold, :);
 end

 function distances = getDistancesMinHashGenres(numFilms,
matrizMinHash,numHash)
      distances = zeros(numFilms,numFilms);
      for n1= 1:numFilms
          for n2= n1+1:numFilms
              distances(n1,n2) = sum(matrizMinHash(:,n1)==
matrizMinHash(:,n2))/numHash;
          end
      end
 end
```

Para além de não termos conseguido implementar por completo a opção 5 no script2.m, está

apenas a verificação dos géneros no input do utilizador e uma tentativa de filtrar as similaridades, mas o resultado não é o esperado.

O código que segue é para o menu presente no script2.m:

```
 case 5
        valid_input = false;
        while ~valid_input
            user_input = input('Select one or more genres (
separated by '',''): ', 's');
            input_genres = strsplit(user_input, ',');
            valid_input = all(ismember(input_genres, genres))
;
            if ~valid_input
                disp('One or more genres are invalid. Please
try again.');
            end
        end
        distancesGenres = getDistancesMinHashGenres(numFilms,
matrizMinHashGenres,numHash);
        user_genre_vector = ismember(genres, input_genres);

        % Sort by similarity and year
        similarGenres = filterSimilarGenres(user_genre_vector
, distancesGenres, 0.9);
        fprintf('%s\n', similarGenres(:, 3));
        [~, sorting_order] = sort(similarGenres(:,3));

        % Display top 5 movies
        fprintf('\nTop 5 similar movies based on selected
genres:\n');
        for i = 1:5
            movie_idx = similarGenres(sorting_order(i), 1);
            movie_year = years(movie_idx);
            movie_title = titles{movie_idx};
            fprintf('%s (%d)\n', movie_title, movie_year);
        end
```

Option 6

Esta opção serve para terminar o programa. Foi feita com o seguinte fragmento de código:

```
 case 6
        return
```

Capítulo 4

Conclusão

Neste trabalho, a nível teórico, consolidámos os nossos conhecimentos sobre a matéria em geral do guião 4 (hash functions, filtros de Bloom ,similaridade, etc). Em suma, quanto ao nível prático, melhorámos não só os nossos conhecimentos sobre Matlab, como também as nossas capacidades de produzir algoritmos probabilísticos ou até mesmo, as nossas capacidades de trabalhar enquanto equipa. Acreditamos ter alcançado grande parte dos objetivos dados pelo professor