Relatório - MPEI PL4

Universidade de Aveiro

Tomás Cerca Rodrigues, Alexandre Costa Martins

Relatório - MPEI PL4

LECI

Universidade de Aveiro

Tomás Cerca Rodrigues, Alexandre Costa Martins
(104090) tcercarodrigues@ua.pt, (103552) alexandremartins@ua.pt

21/12/2023

Índice

Capítulo 1

Introdução

Este trabalho consistiu no desenvolvimento duma aplicação em MATLAB para simular um sistema de informação para pesquisa de filmes.

Foi entregue aos alunos o ficheiro movies.csv com uma lista de 58000 filmes existentes e requesitado que a aplicação futuramente desenvolvida aceda *às cells* e apresente 5 opções.

1. Apresentar todos os géneros disponíveis.
2. Número de filmes de um determinado genero, sendo este recebido por input do usuário.
3. Número de filmes de um determinado genero e ano, sendo estes recebidos também por input do usuário.
4. Pesquisar filmes pelo título.
5. Pesquisar filmes baseado no genero.

Ao longo deste relatório serão fundamentadas as opções tomadas pelo grupo na implementação dos diversos métodos probabilisticos recorridos para a resolução das atividades.

Capítulo 2

Interface e Main

2.1 main.m

```
%% main
movies = readcell('movies.csv', 'Delimiter',',');
option = interfaceSelecao();

while (option ~= 6)

    switch option
        case 1
            fprintf('Option 1 selected...\n');
            % 'Display avaliable genres'
            genres = getGenres(movies);
            disp('All movie genres:');
            disp(genres);

        case 2
            fprintf('Option 2 selected...\n');
            % 'Number of movies of a genre'
            fprintf('Executing option 2 ...\n');
            genre = input('Enter the genre: ', 's');
            count = countMoviesByGenre(movies, genre);

        case 3
            fprintf('Option 3 selected...\n');
            % 'Number of movies of a genre on a given
               year's
            genre = input('Enter the genre: ', 's');
            year = input('Enter the year: ');
```

```
27              count = countMoviesByGenreAndYear(movies,
                    genre,year);
28
29          case 4
30              fprintf('Option 4 selected...\n');
31              % 'Search of movie titles'
32              title = input('Enter the title: ', 's');
33              similarMoviesByTitle = getMoviesByTitle(
                    movies, title);
34              disp('All similar movie titles:');
35              disp(similarMoviesByTitle);
36
37          case 5
38              fprintf('Option 5 selected...\n');
39              % 'Search movies based on genres'
40              genres = input('Enter the genres (
                    separated by commas): ', 's');
41              genres = strsplit(genres, ',');
42              similarMoviesByGenre = getMoviesByGenre(
                    movies, genres);
43              disp('All movie with the desired genre: ')
                    ;
44              disp(similarMoviesByGenre);
45
46          case 6
47              fprintf('Closing the program...\n');
48              % 'Exit'
49              return;
50
51          otherwise
52              fprintf('Invalid option selected.\n');
53      end
54
55      option = interfaceSelecao();
56  end
```

O main é o script que corre o programa, é neste mesmo script que são chamadas todas as funções desenvolvidas não só referentes às funcionalidades da aplicação como referentes até à interface.

2.2 interfaceSelecao.m

```
function numeroSelecionado = interfaceSelecao()
opcoes = {'Display avaliable genres',
        'Number of movies of a genre',
        'Number of movies of a genre on a given year',
        'Search of movie titles',
        'Search movies based on genres',
        'Exit'}; % Option list


while true
    % Show enumerated options
    for i = 1:numel(opcoes)
        fprintf('%d. %s\n', i, opcoes{i});
    end

    % Gets input
    numeroSelecionado = input('Input your desired
       option: ');

    if  ((numeroSelecionado >= 1) && (
       numeroSelecionado<= 6))
        fprintf('Valid option\n ');
        return;

    end
end
```

O seguinte script é meramente responsável pela interação do usuário com o menu da aplicação, expondo as opções disponíveis, adquirindo e por fim validando os valores de entrada.

Capítulo 3

Opção 1

3.1 getGenres.m

```
1  function genreArray = getGenres(movies)
2
3      maxGenres = size(movies, 1) * (12 - 3 + 1);
4      genres = cell(1, maxGenres);
5      genreCount = 0;
6
7      for i = 1:size(movies, 1)
8          for j = 3:12
9              if numel(movies{i,j}) > 1 && ~strcmp(
                   movies{i,j}, '(no genres listed)')
10                 genreCount = genreCount + 1;
11                 genres{genreCount} = movies{i, j};
12                 movies{i,j};
13             end
14         end
15     end
16
17     genres = genres(1:genreCount);
18
19     genreArray = unique(genres);
20 end
```

Esta função é chamada pela main quando o usuário seleciona a opção 1 do menu e tem como objetivo listar todos os generos de filmes presentes no ficheiro e na cell *movie.csv*.

O único parâmetro de entrada são as *cells* de *movie.csv*.

Capítulo 4

Opção 2

4.1 countMoviesByGenre.m

```
1  function estimatedCount = countMoviesByGenre(movies,
       userGenre)
2      movies = movies(:, :);
3      userGenre = lower(userGenre);
4
5      maxGenres = size(movies, 1) * (12 - 3 + 1);
6      genres = cell(1, maxGenres);
7      genreCount = 0;
8
9      for i = 1:size(movies, 1)
10         for j = 3:12
11
12             if numel(movies{i,j}) > 1 && ~strcmp(
                   movies{i,j}, '(no genres listed)')
13                 genreCount = genreCount + 1;
14                 genres{genreCount} = lower(movies{i, j
                       });
15             end
16         end
17     end
18
19
20     n = 1000;
21     bloomFilter = inicializar(n);
22
23     k = 3;
24     for i = 1:size(genres, 1)
25         for j = 1:size(genres, 2)
```

```
26              genre = genres{i, j};
27              if ~isempty(genre)
28                  bloomFilter = adicionar_elemento(
                        bloomFilter, genre, k);
29              end
30          end
31      end
32
33      if pertenca(bloomFilter, userGenre, k)
34          disp(The genre is present in the Bloom Filter
                .');
35
36          estimatedCount = 0;
37          for i = 1:size(genres, 1)
38              for j = 1:size(genres, 2)
39                  genre = genres{i, j};
40                  if strcmp(userGenre, genre)
41                      estimatedCount = estimatedCount +
                            1;
42                  end
43              end
44          end
45           disp(['Estimated number of movies ',
                userGenre, ': ', num2str(estimatedCount)])
                ;
46      else
47          disp('The genre is not present in the Bloom
                Filter');
48      end
49
50
51  end
```

Como já sugere o nome, este script é o principal responsável pela contagem/estimativa de filmes de um determinado genero. No começo do script são inicializados e preparados todos os dados e varíaveis, ressaltamos "*movies = movies(:, :)*"que garante que serão abordadas todas as colunas da cell e "*userGenre = lower(userGenre)*"que torna menos rigída a compreensão de inputs por parte da aplicação parando de diferenciar letras maiúsculas de minúsculas.

Na criação do filtro de Bloom inicializamos **n**, o número de *bits* do filtro a 1000 e **k**, o número de funções de dispersão a 3.

Após o seu preenchimento verificamos se o genero fornecido pelo usuário se encontra presente no filtro, caso esteja presente o filtro itá calcular e exibir uma estimativa do número de filmes que contêm esse genero. Caso este esteja ausente será exibida uma mensagem ao utilizando relatando esse mesmo caso.

Capítulo 5

Opção 3

5.1 countMoviesByGenreAndYear.m

```
function estimatedCount = countMoviesByGenreAndYear(movies, genre, year)
    genre = strtrim(lower(genre));
    year = strtrim(num2str(year));

    n = 1000;
    bloomFilter = inicializar(n);

    maxGenres = size(movies, 1) * (12 - 3 + 1);
    genres = cell(1, maxGenres);
    genreCount = 0;

    for i = 1:size(movies, 1)
        for j = 3:12
            if numel(movies{i,j}) > 1 && ~strcmp(movies{i,j}, '(no genres listed)')
                genreCount = genreCount + 1;
                genres{genreCount} = lower(movies{i, j});

                k = 3;
                bloomFilter = adicionar_elemento(bloomFilter, genres{genreCount}, k);
            end
        end
    end

```

```
    % check if the genre is in the bloom filter
    if pertenca(bloomFilter, genre, k)
        disp(['The genre ', genre, ' is present in the
            Bloom Filter.']);

        estimatedCount = 0;
        for i = 1:size(movies, 1)
            movieYear = strtrim(num2str(movies{i, 2}))
                ;
            movieGenres = {};
            for j = 3:12
                if ~ismissing(movies{i, j})
                    movieGenres = [movieGenres,
                        strsplit(lower(movies{i, j}), '
                        |')];
                end
            end

            if any(cellfun(@(x) any(strcmp(genre, x)),
                movieGenres)) && strcmp(year,
                movieYear)
                estimatedCount = estimatedCount + 1;
            end
        end
        disp(['Estimated number of movies of genre ',
            genre, ' and year ', year, ': ', num2str(
            estimatedCount)]);
    else
        disp(['The genre ', genre, ' is not present in
            the Bloom Filter.']);
        estimatedCount = 0;
    end
end
```

Este cript por mais que seja referente a outro exercício, os métodos de resolução solicitados pelo guião são os mesmos o que faz com que a sua estrutura e desenvolvimento sejam extremamente semelhantes. Contude tem leves extras que valem ser apontados.

Pra receber os valores do inputs o ano é passado a String e o genero é novamente convertido para minúsculas pelo motivo já mencionado.

O filtro de Bloom é iniciado e é verificado a presença do genero neste, caso estes estejam presentes é iniciada e depois exibida a contagem referente aos filmes desse mesmo genero e ano. Caso o genero não esteja presente o comportamento da aplicação é o mesmo referido no capítulo anterior, ou seja, será exibida uma mensagem informando que o genero não se encontra presente no filtro.

Capítulo 6

Opção 4

6.1 getMoviesByTitle.m

```
function titleArray = getMoviesByTitle(movies, title)

    title = lower(title);
    maxMovies = size(movies, 1);
    titleArray = cell(maxMovies, 2);
    movieCount = 0;

    shingleSize = 2;
    numHashFuncs = 100;

    titleShingles = createShingles(title, shingleSize)
        ;
    titleSignature = minHash(titleShingles,
        numHashFuncs);

    for i = 1:size(movies, 1)
        movieTitle = lower(movies{i, 1});
        movieShingles = createShingles(movieTitle,
            shingleSize);
        movieSignature = minHash(movieShingles,
            numHashFuncs);

        similarity = sum(titleSignature ==
            movieSignature) / numHashFuncs;

        if similarity > 0.6
            movieCount = movieCount + 1;
            titleArray{movieCount, 1} = movies{i, 1};
```

```
24              titleArray{movieCount, 2} = similarity;
25          end
26      end
27      titleArray = titleArray(1:movieCount, :);
28      titleTable = cell2table(titleArray, 'VariableNames
            ', {'Title', 'Similarity'});
29      titleTable = sortrows(titleTable, 'Similarity', '
            descend');
30      titleArray = table2cell(titleTable);
31  end
32
33  function shingles = createShingles(str, k)
34      shingles = arrayfun(@(i) str(i:i+k-1), 1:length(
            str)-k+1, 'UniformOutput', false);
35  end
36
37  function signature = minHash(shingles, numHashFuncs)
38      signature = inf(1, numHashFuncs);
39      for i = 1:length(shingles)
40          hashValues = mod(sum(double(shingles{i})) .*
                (1:numHashFuncs), numHashFuncs);
41          signature = min(signature, hashValues);
42      end
43  end
```

Esta função recebe apenas a lista de filmes e um título para procurar, após a conversão deste para minúsculas.

Inicializa depois variáveis para armazenar títulos semelhantes encontrados e depois define o tamanho de shingle para 2 e o número de funções hash para 100. Usámos shingle a 2 para poder ser razoavelmente mais exigente nasprocura e obter filmes com o títulos similares. Caso shingle fosse 3 ser-nos-ia devolvido títulos que quase não teriam similaridade com o input do utilizador.

Usamos também minHash com os shingles para poder criar uma assinatura minHash.

O processo de comparação, novamente, converte o título para minusculas, gera shingles para este, calcula a assinatura minHash e a similaridade. Se a similaridade for maior que 0.6 este título é adicionado à lista de filmes encontrados.

Capítulo 7

Opção 5

7.1 getMoviesByGenre.m

```
function genreArray = getMoviesByGenre(movies,
    inputGenres)
    if ischar(inputGenres)
        inputGenres = cellstr(inputGenres);
    end
    if isstring(inputGenres)
        inputGenres = {inputGenres};
    end

    maxMovies = size(movies, 1);
    genreArray = cell(maxMovies, 3);
    movieCount = 0;

    shingleSize = 2;
    numHashFuncs = 100;

    h = waitbar(0,'Please wait...');


    for i = 1:size(movies, 1)
        waitbar(i/maxMovies, h);
        genreMatchCounter = 0;
        for j = 3:12
            if numel(movies{i,j}) > 1 && ~strcmp(
                movies{i,j}, '(no genres listed)')
                movieGenre = lower(movies{i, j});
                movieShingles = createShingles(
                    movieGenre, shingleSize);
```

```
26                  movieSignature = minHash(movieShingles
                        , numHashFuncs);
27
28
29                  for g = 1:length(inputGenres)
30                      genre = lower(strtrim(inputGenres{
                            g}));
31                      genreShingles = createShingles(
                            genre, shingleSize);
32                      genreSignature = minHash(
                            genreShingles, numHashFuncs);
33
34                      similarity = sum(genreSignature ==
                             movieSignature) / numHashFuncs
                            ;
35
36                      if similarity > 0.6
37                          genreMatchCounter =
                                genreMatchCounter + 1;
38                      end
39                  end
40              end
41          end
42          finalSimilarity = genreMatchCounter / length(
                inputGenres);
43          if finalSimilarity > 0.6
44              movieCount = movieCount + 1;
45              genreArray{movieCount, 1} = movies{i, 1};
46              genreArray{movieCount, 2} = movies{i, 2};
47              genreArray{movieCount, 3} =
                    finalSimilarity;
48          end
49      end
50
51      close(h); % close waitbar
52
53      genreArray = genreArray(1:movieCount, :);
54      genreTable = cell2table(genreArray, 'VariableNames
            ', {'Title', 'Year', 'Similarity'});
55      genreTable = sortrows(genreTable, {'Similarity', '
            Year'}, {'descend', 'descend'});
56      if height(genreTable) > 50
57          genreTable = genreTable(1:50, :);
58      end
59      genreArray = table2cell(genreTable);
60  end
```

```
61
62
63 function shingles = createShingles(str, k)
64     shingles = arrayfun(@(i) str(i:i+k-1), 1:length(
           str)-k+1, 'UniformOutput', false);
65 end
66
67
68 function signature = minHash(shingles, numHashFuncs)
69     signature = inf(1, numHashFuncs);
70     for i = 1:length(shingles)
71         hashValues = mod(sum(double(shingles{i})) .*
               (1:numHashFuncs), numHashFuncs);
72         signature = min(signature, hashValues);
73     end
74 end
```

Este é o script da opção final e tem algumas características já implementadas nos exercícios anteriores (principalmente a opção 4). Tal como os demais, este recebe a lista de filmes e agora um ou mais generos separados por uma vírgula.

O processo de comparação, como esperado, é a principal diferença ainda que esta não seja muito grande. Neste são enunciados os generos listadosno filme, de seguida é feita a comparação de assinaturas minHash dos generos introduzidos pelo utilizador com os generos dos filmes.

Novamente a similaridade desejada foi 0.6, ou seja todos os filmes com uma similaridade superior eram adicionados à lista.

A tabela final é ordenada por ordem decrascente de similaridade e o número de resultados é limitado a 50 caso a função queira devolver mais que isso.