deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

PL 4
Algoritmos Probabilísticos

Métodos Probabilísticos para Engenharia Informática

Trabalho realizado por:

Tomás Sousa Fonseca, nº 107245, P3

Beatriz Filipa Da Silva Ferreira, nº 107214, P3

Ano Letivo 2023/2024

Índice

Introdução 2
Desenvolvimento da Aplicação 3
createWorkspace.m 3
Menu.m 4
Opções Desenvolvidas 6
    Opção 1 6
    Opção 2 9
    Opção 3 13
    Opção 4 17
    Opção 5 21
Conclusão 25

Introdução

Este trabalho é realizado no âmbito da disciplina de Métodos Probabilísticos para Engenharia Informática do 3º ano da Licenciatura em Engenharia de Computadores e Informática.

Foi-nos proposto desenvolver uma aplicação, em MATLAB, com algumas funcionalidades de um sistema de informação para pesquisa de filmes. Esta aplicação vai fazer uso de 1 ficheiro previamente disponibilizado, “movies.csv”, de onde iremos retirar uma lista com 58000 filmes existentes.

Com isto em mente, temos como objetivo principal realizar uma aplicação totalmente funcional, com todas as opções do menu operacionais.

Desenvolvimento da Aplicação

Numa primeira fase, criámos dois scripts, o primeiro corre uma única vez para ler o ficheiro de entrada e guardar num ficheiro em disco todos os dados necessários à implementação da aplicação – **createWorkspace.m** – e o segundo para implementar a aplicação em si – **menu.m**. Esta separação da informação simplifica a execução da aplicação, uma vez que facilita o carregamento dos dados necessários para o script da aplicação.

createWorkspace.m

A função createWorkspace, desenvolvida em MATLAB, tem como finalidade facilitar a manipulação de dados relacionados aos filmes.

```
function createWorkspace(moviesFile, workspaceFile)

    movies = readcell(moviesFile, 'Delimiter', ','); %#ok<*NASGU>


    shingleSize = 4;
    numHashFunctions = 3;


    save(workspaceFile);
end
```

Figura 1 - Função createWorkspace.m

A função inicia com a leitura de dados provenientes do ficheiro **moviesFile.** Utiliza-se a função **readcell** para interpretar o conteúdo do ficheiro, considerando que os valores estão separados por vírgulas (**','**). Os dados lidos são armazenados na variável **movies**.

Definimos dois parâmetros:

shingleSize = 4;

A variável **shingleSize** foi definida como 4, o que implica que subsequências de 4 caracteres serão utilizadas em algum ponto do código, neste caso para calcular similaridade entre strings.

numHashFunctions = 3;

Por último, salvamos o estado atual do espaço de trabalho num ficheiro, cujo nome é especificado por **workspaceFile**. Isso significa que as variáveis **movies**, **shingleSize** e **numHashFunctions**, serão salvas para uso futuro.

Menu.m

No ficheiro "Menu.m", apresentamos uma implementação interativa para análise de dados relacionados a filmes. O menu oferece diversas opções, desde a visualização de géneros disponíveis até a pesquisa de títulos e filmes com base em géneros específicos.

Antes de apresentar o menu principal, o código realiza um pré-processamento dos dados. O ficheiro workspace.mat é verificado, e se não existir, os dados são lidos a partir do ficheiro movies.csv utilizando a função createWorkspace. Além disso, strings vazias e a string especial '(no genres listed)' são tratadas, visando garantir a consistência e qualidade dos dados.

No menu principal, temos 6 opções para o utilizador e solicitamos que o utilizador escolha uma opção válida e verificamos a validade da opção inserida.

```
workspaceFile = 'workspace.mat';
moviesFile = 'movies.csv';

if exist(workspaceFile, 'file') ~= 2
    fprintf('Creating Workspace...\n');
    createWorkspace(moviesFile, workspaceFile);
end

fprintf('Loading Workspace...\n');
load(workspaceFile);

% Substituir strings vazias e '(no genres listed)'
movies(cellfun(@(x) isempty(x) || strcmp(x, '(no genres listed)'), movies)) = {''};

while true
    % Menu
    fprintf('Select an option:\n')
    fprintf('1 - Display available genres\n')
    fprintf('2 - Number of movies of a genre\n')
    fprintf('3 - Number of movies of a genre on a given year\n')
    fprintf('4 - Search movie titles\n')
    fprintf('5 - Search movies based on genres\n')
    fprintf('6 - Exit\n')
    option = input('Select an option: ');
    while (option<1 || option>6)
        disp('ERROR! Select a valid option!')
        option = input('Select an option: ');
    end
```

Figura 2 - Função Menu.m (sem os casos)

Após o pré-processamento dos dados e apresentação do menu principal, a aplicação oferece um conjunto de funcionalidades para a exploração dos dados de filmes. De seguida, serão detalhadas as opções disponíveis.

Opções Desenvolvidas

Opção 1

No contexto da aplicação de análise de dados de filmes, a opção 1 foi introduzida para fornecer ao utilizador uma visão detalhada dos géneros presentes nos dados.

Ao escolher a opção 1 no menu, a aplicação executa uma série de operações específicas. A seguir, apresentamos a implementação do case 1 no menu e uma explicação passo a passo dessa secção do código.

```
case 1
    display_available_genres(movies, 1);
    disp('Saving...')
    save(workspaceFile, 'movies');
```

Figura 3 - case 1

1. A linha **display_available_genres(movies, 1)**; invoca a função **display_available_genres** com dois argumentos: o conjunto de dados movies e o valor 1 para o parâmetro option. Esta função é responsável por identificar e apresentar os géneros existentes nos filmes.

2. A linha **disp('Saving...')** exibe no ecrã a mensagem 'Saving...'.

3. A linha **save(workspaceFile, 'movies')**; utiliza a função save para guardar o estado atual dos dados de filmes no ficheiro workspace.mat.

A função **display_available_genres** é responsável por apresentar os géneros únicos existentes nos dados dos filmes, sendo chamada no case 1 do menu quando o utilizador escolhe visualizar os géneros disponíveis. Vamos analisar cada parte da função:

```
function genres_unique = display_available_genres(movies, option)
    % Exibir gêneros para todos os filmes
    fprintf('Genres for all movies:\n');
    genres_unique = [];
    for i = 1:size(movies, 1)
        genres = movies(i, 3:end);
        for j = 1:length(genres)
            if ~ismissing(genres{j})
                if ~ismember(genres{j}, genres_unique)
                    genres_unique = [genres_unique , cellstr(genres{j})];
                end
            end
        end
    end
    switch option
        case 1
            for f = 1:length(genres_unique)
                if f == 19
                    disp(genres_unique{f})
                    fprintf('\n')
                else
                    if f == 10
                        fprintf('\n')
                    end
                    fprintf('%s%s', genres_unique{f}, ' \ ')
                end
            end
        case 2
            for f = 1:length(genres_unique)
                if f == 19
                    fprintf('%d - %s\n', f, genres_unique{f})
                    fprintf('\n')
                else
                    if f == 10
                        fprintf('\n')
                    end
                    fprintf('%d - %s%s', f, genres_unique{f}, ' \ ')
                end
            end
    end
```

Figura 4 - Função display_available_genres.m

1. A função recebe como entrada o conjunto de dados **movies** e o parâmetro **option**, que é utilizado para personalizar o formato da apresentação dos géneros.

2. Inicializa um vetor vazio chamado **genres_unique** que será usado para armazenar os géneros únicos encontrados nos dados dos filmes.

3. Utiliza dois loops aninhados para percorrer cada filme e cada género associado a esse filme e adiciona géneros únicos ao vetor **genres_unique** evitando duplicados.

4. Utiliza uma estrutura **switch** para determinar como os géneros serão apresentados, dependendo do valor de **option**.

5. Se **option** é 1, apresenta os géneros de forma linear, com quebras de linha para melhor legibilidade.

6. Se **option** é 2, apresenta os géneros de forma linear, mas inclui números de identificação.

Opção 2

No contexto da aplicação de análise de dados de filmes, a opção 2 foi introduzida para fornecer ao utilizador o número de filmes classificados no género que o utilizador seleciona antecipadamente.

Ao escolher a opção 2 no menu, a aplicação executa uma série de operações específicas. A seguir, apresentamos a implementação do case 2 no menu e uma explicação passo a passo dessa secção do código.

```
case 2
    genres_unique = display_available_genres(movies, 2);
    genre = input('Select a genre: ');
    while (genre<=0 || genre > length(genres_unique))
        disp('ERROR! Select a valid option!')
        genre = input('Select a genre: ');
    end
    movies_of_genre(movies, genres_unique{genre});
```

Figura 5 - case 2

1. Chama a função **display_available_genres** com **movies** e **2** como argumentos. Essa função apresenta os géneros de forma numérica e retorna a lista de géneros únicos.
2. Solicita ao utilizador selecionar um género digitando um número associado ao género desejado.
3. Entra em um loop enquanto o utilizador selecionar um número fora do intervalo válido (número de géneros disponíveis).
    a. Se o utilizador selecionar um número inválido, exibe uma mensagem de erro.
    b. Solicita ao utilizador fazer uma nova seleção de género.
4. Quando o utilizador seleciona um género válido, chama a função **movies_of_genre** passando o conjunto de dados **movies** e o género selecionado pelo utilizador.

A função **movies_of_genre** é responsável por indicar o número de filmes classificados no género selecionado sendo chamada no case 2 do menu. Vamos analisar cada parte da função:

```
function count = movies_of_genre(movies, genre)

    hashes = collect_hashes(movies, 2, 0);

    % Create containers.Map with string keys
    bloomFilter = containers.Map('KeyType','char','ValueType','uint32');

    % Count occurrences of hashes using the containers.Map
    for i = 1:numel(hashes)
        hashKey = hashes{i};
        if isKey(bloomFilter, hashKey)
            bloomFilter(hashKey) = bloomFilter(hashKey) + 1;
        else
            bloomFilter(hashKey) = 1;
        end
    end

    % Count number of movies for the selected genre
    genreHash = string2hash(genre);
    genreHashStr = num2str(genreHash);
    count = bloomFilter(genreHashStr);

    disp(['Number of movies in genre "', genre, '": ', num2str(count)]);

    fprintf('\n')
end
```

Figura 6 - Função movies_of_genre.m

1. Chama a função **collect_hashes** para gerar hashes dos títulos dos filmes (com shingle size 2) e armazená-los no vetor **hashes**

2. Inicializa um objeto containers.Map para contar as ocorrências dos hashes

3. Itera sobre o vetor de hashes gerados

    a. Obtém o valor do hash na posição atual
    b. Verifica se o hash já está presente no **containers.Map**
    c. Incrementa a contagem se o hash já estiver presente
    d. Adiciona o hash ao **containers.Map** se ainda não estiver presente

4. Converte o género selecionado para um valor de hash

5. Converte o valor de hash para uma string

6. Obtém a contagem de filmes para o género selecionado do **containers.Map**

7. Apresenta no ecrã o número de filmes associados ao género selecionado

8. Adiciona uma linha em branco para melhor formatação

A função **collect_hashes** é responsável por gerar hashes a partir dos géneros dos filmes presentes nos dados, de acordo com a opção escolhida sendo chamada na função **movies_of_genre**.

```
      function hashes=collect_hashes(movies, option, year)
    % Array para guardar as hashes
    hashes = {};

    yearsData = cell2mat(movies(:, 2));
    switch option

        case 2
            % Loop through each cell to collect hashes
            for i = 1:size(movies, 1)
                for j = 1:size(movies, 2)
                    currentGenre = movies{i, j};
                    if ischar(currentGenre) && ~isempty(currentGenre) && ~strcmp(currentGenre, "missing")
                        % Use string hashing function to generate hash
                        hashValue = string2hash(currentGenre);
                        % Convert hash to string before storing
                        hashStr = num2str(hashValue);
                        hashes = [hashes, hashStr];
                    elseif iscell(currentGenre)
                        for k = 1:numel(currentGenre)
                            if ischar(currentGenre{k}) && ~isempty(currentGenre{k}) &&
~strcmp(currentGenre{k}, "missing")
                                % Use string hashing function to generate hash
                                hashValue = string2hash(currentGenre{k});
                                % Convert hash to string before storing
                                hashStr = num2str(hashValue);
                                hashes = [hashes, hashStr];
                            end
                        end
                    end
                end

                progress = (i*100)/length(movies);
                fprintf('Loading Results...%.2f %%\n', progress);
            end

        case 3
            movies = movies(:, 3:end);
            % Loop through each cell to collect hashes
            for i = 1:size(movies, 1)
                if yearsData(i) == year
                    for j = 1:size(movies, 2)
                    currentGenre = movies{i, j};
                    if ischar(currentGenre) && ~isempty(currentGenre) && ~strcmp(currentGenre, "missing")
                        % Use string hashing function to generate hash
                        hashValue = string2hash(currentGenre);
                        % Convert hash to string before storing
                        hashStr = num2str(hashValue);
                        hashes = [hashes, hashStr];
                    elseif iscell(currentGenre)
                        for k = 1:numel(currentGenre)
                            if ischar(currentGenre{k}) && ~isempty(currentGenre{k}) &&
~strcmp(currentGenre{k}, "missing")
                                % Use string hashing function to generate hash
                                hashValue = string2hash(currentGenre{k});
                                % Convert hash to string before storing
                                hashStr = num2str(hashValue);
                                hashes = [hashes, hashStr];
                            end
                        end
                    end
                end
                end

                progress = (i*100)/length(movies);
                fprintf('Loading Results...%.2f %%\n', progress);
            end
    end
    fprintf('\n');
end
```

1. **hashes** = {}: Inicializa um vetor vazio para armazenar as hashes geradas
2. Utiliza uma estrutura **switch** para determinar a ação a ser tomada com base na opção fornecida
3. Para cada filme e género nos dados:
    a. Verifica se o género é uma string não vazia e não é "missing"
    b. Gera uma hash para o género usando a função **string2hash**
    c. Armazena a hash como uma string no vetor **hashes**
4. Exibe o progresso do processo de carregamento em percentagem.
5. Retorna o vetor **hashes** contendo todas as hashes geradas.

Detalhes por Opção:

- **Opção 2:**
    - Gera hashes para todos os géneros presentes nos dados.
- **Opção 3:**
    - Gera hashes apenas para os géneros dos filmes de um ano específico.

Opção 3

Na aplicação de análise de dados de filmes, a opção 3 foi introduzida para fornecer ao utilizador o número de filmes classificados em um género e um ano específico que o utilizador seleciona antecipadamente. A seguir, apresentamos a implementação do case 3 no menu e uma explicação passo a passo dessa secção do código.

```
case 3
    bool = true;
    while bool
        genres_unique = display_available_genres(movies, 2);
        user_input = input('Select a genre, followed by a year (separated by ","): ', 's');
        data = strsplit(user_input, ',');
        if length(data) ~= 2  %% verifica se o input contem 2 argumentos
            disp('ERROR! Not enough arguments!')
        else
            genre=str2double(data{1});
            year=str2double(data{2});
            length_year = floor(log10(abs(year))) + 1;
            if genre<=0 || genre > length(genres_unique)
                disp('ERROR! Wrong genre input!')
            elseif year>2023 || length_year ~= 4
                disp('ERROR! Wrong year input!')
            else
                bool = false;
            end
        end
    end
    movies_of_genre_year(movies, genres_unique{genre}, year);
```

Figura 7 - case 3

1. **Inicialização do Loop e Obtenção de Género e Ano:**

    - **bool = true;**: Inicializa uma variável booleana para controlar o loop.
    - **while bool**: Entra num loop enquanto **bool** for verdadeiro.
    - **genres_unique = display_available_genres(movies, 2);**: Chama a função **display_available_genres** para obter a lista de géneros únicos.
    - **user_input = input('Select a genre, followed by a year (separated by ","): ', 's');**: Solicita ao utilizador um género e um ano separados por vírgula.

2. **Validação da Entrada do Utilizador:**

    - **data = strsplit(user_input, ',');**: Divide a entrada do utilizador nas partes de género e ano.
    - **if length(data) ~= 2**: Verifica se a entrada contém exatamente dois argumentos.

- **disp('ERROR! Not enough arguments!')**: Exibe uma mensagem de erro se a entrada for inválida.

- **else**: Se a entrada for válida:
  - **genre=str2double(data{1});**: Converte o género para um número.
  - **year=str2double(data{2});**: Converte o ano para um número.
  - **length_year = floor(log10(abs(year))) + 1;**: Calcula o comprimento do ano.
  - Valida se o género está dentro do intervalo e se o ano é válido.

3. **Chamada da Função movies_of_genre_year:**

   - **movies_of_genre_year(movies, genres_unique{genre}, year);**: Quando o utilizador fornece uma entrada válida, chama a função **movies_of_genre_year** passando os dados **movies**, o género selecionado e o ano selecionado.

```
function count = movies_of_genre_year(movies, genre, year)

    hashes = collect_hashes(movies, 3, year);

    % Create containers.Map with string keys
    bloomFilter = containers.Map('KeyType','char','ValueType','uint32');

    % Count occurrences of hashes using the containers.Map
    for i = 1:numel(hashes)
        hashKey = hashes{i};
        if isKey(bloomFilter, hashKey)
            bloomFilter(hashKey) = bloomFilter(hashKey) + 1;
        else
            bloomFilter(hashKey) = 1;
        end
    end

    % Count number of movies for the selected genre in the given year
    genreHash = string2hash(genre);
    genreHashStr = num2str(genreHash);
    count = bloomFilter(genreHashStr);

    fprintf('Number of movies in genre "%s" in year %d: %d\n', genre, year, count);
    fprintf('\n');
end
```

Figura 8 - Função movies_of_genre_year

A função **movies_of_genre_year** é responsável por indicar o número de filmes do ano e género selecionado sendo chamada no case 3 do menu. Vamos analisar cada parte da função:

1. **Geração de Hashes para o Ano Específico:**

    - **hashes = collect_hashes(movies, 3, year);**: Chama a função **collect_hashes** para gerar hashes dos géneros dos filmes especificamente para o ano fornecido.

2. **Inicialização de um containers.Map:**

    - **bloomFilter = containers.Map('KeyType','char','ValueType','uint32');**: Inicializa um objeto **containers.Map** para contar as ocorrências dos hashes.

3. **Contagem de Ocorrências de Hashes:**

    - **for i = 1:numel(hashes)**: Itera sobre o vetor de hashes gerados.
        - **hashKey = hashes{i};**: Obtém o valor do hash na posição atual.
        - **if isKey(bloomFilter, hashKey)**: Verifica se o hash já está presente no **containers.Map**.
            - **bloomFilter(hashKey) = bloomFilter(hashKey) + 1;**: Incrementa a contagem se o hash já estiver presente.
            - **else**: Se o hash não estiver presente:
                - **bloomFilter(hashKey) = 1;**: Adiciona o hash ao **containers.Map** com uma contagem de 1.

4. **Conversão do Género Selecionado para Hash e Contagem:**

    - **genreHash = string2hash(genre);**: Converte o género selecionado para um valor de hash.

- **genreHashStr = num2str(genreHash);**: Converte o valor de hash para uma string.
- **count = bloomFilter(genreHashStr);**: Obtém a contagem de filmes para o género selecionado do **containers.Map**.

5. **Apresentação dos Resultados:**
    - **fprintf('Number of movies in genre "%s" in year %d: %d\n', genre, year, count);**: Apresenta no ecrã o número de filmes associados ao género selecionado no ano especificado.
    - **fprintf('\n');**: Adiciona uma linha em branco para melhor formatação.

Opção 4

No contexto da aplicação de análise de dados de filmes, a opção 4 foi introduzida para permitir que o utilizador pesquise títulos de filmes com base em uma string inserida. Ao escolher a opção 4 no menu, a aplicação executa uma série de operações específicas. A seguir, apresentamos a implementação do case 4 no menu e uma explicação passo a passo dessa secção do código.

```
case 4
    user_input = input('Insert a string: ','s');
    search_movie_title(movies, user_input,shingle_size);
```

Figura 9 – case 4

1. Solicita ao utilizador inserir uma string que será usada como base para a pesquisa de títulos de filmes.

2. Converte a string de input para minúsculas e remove espaços em branco.

3. Chama a função **search_movie_title** passando os dados dos filmes (**movies**), a string inserida pelo utilizador (**input_string**), e o tamanho dos shingles (**shingle_size**).

A função **search_movie_title** é responsável por encontrar os filmes mais semelhantes à string inserida pelo utilizador. Vamos analisar cada parte da função:

```
function search_movie_title(movies, input_string,shingle_size)
    movie_names = movies(:, 1);
    movie_genres = movies(:, 3:end);

    % Criar shingles para a entrada do usuário
    input_shingles = generate_shingles(input_string, shingle_size);

    jaccard_indices = zeros(length(movie_names), 1);
    for i = 1:length(movie_names)
        % Calcular o índice de Jaccard entre a entrada do usuário e o nome do filme
        movie_shingles = generate_shingles(char(movie_names(i)), shingle_size);

        % Verificar se as strings são exatamente iguais (sem distinção entre maiúsculas e minúsculas)
        intersection = length(intersect(input_shingles, movie_shingles));
        unionSet = length(union(input_shingles, movie_shingles));
        jaccard_index = intersection / unionSet;


        % Atribuir o índice de Jaccard ao vetor
        jaccard_indices(i) = jaccard_index;

        progress = (i*100)/length(movies);
        fprintf('Loading Results...%.2f %%\n', progress);
    end

    [~, indices] = sort(jaccard_indices, 'descend');

    fprintf('Top 5 Movies Similar to "%s":\n', input_string);
    count = 0;
    for i = 1:length(indices)
        movie_index = indices(i);
        % Verificar se o índice de Jaccard é maior que zero (há alguma semelhança)
        if jaccard_indices(movie_index) > 0
            count = count + 1;
            genres = strjoin(cellstr(movie_genres{movie_index}), '\t');
            fprintf('Movie: %-20s Genres: %-20s Jaccard Indice: %f\n', movie_names{movie_index}, genres, jaccard_indices(movie_index));
        end
        if count == 5
            break; % Parar após encontrar os top 5 filmes correspondentes
        end
    end
    fprintf('\n');
end
```

Figura 10 – Função search_movie_title

1. **Obtenção dos Dados dos Filmes:**
    - **movie_names = movies(:, 1);**: Obtém os nomes dos filmes da primeira coluna dos dados dos filmes.
    - **movie_genres = movies(:, 3:end);**: Obtém os géneros dos filmes das colunas 3 até a última.
2. **Criação de Shingles para a Entrada do Utilizador:**
    - **input_shingles = generate_shingles(input_string, shingle_size);**: Converte a string de entrada do usuário em shingles (sequências curtas de caracteres) usando a função **generate_shingles**.
3. **Cálculo dos Índices de Jaccard:**
    - Para cada filme nos dados:
        - **movie_shingles = generate_shingles(char(movie_names(i)), shingle_size);**: Converte o nome do filme em shingles.

- Calcula o índice de Jaccard, que mede a similaridade entre os shingles da entrada do usuário e os shingles do filme.

4. **Ordenação dos Filmes por Similaridade:**
    - Os índices de Jaccard são ordenados em ordem descendente para identificar os filmes mais semelhantes primeiro.
5. **Apresentação dos Resultados - Top 5 Filmes Mais Semelhantes:**
    - Itera sobre os índices ordenados.
    - **genres = strjoin(cellstr(movie_genres{movie_index}), '\t');**: Converte os géneros do filme em uma string formatada, separando-os por tabulação.
    - Apresenta na saída o nome do filme, os géneros associados e o índice de Jaccard.
    - O processo é interrompido após apresentar os top 5 filmes mais semelhantes.
6. **Exibição do Progresso:**
    - Durante o processo, exibe o progresso em percentagem para fornecer feedback visual ao usuário.

A função **generate_shingles** é responsável por converter uma string de entrada em shingles. Vamos analisar cada parte da função:

```
function shingles = generate_shingles(input_string, shingle_size)
    shingles = {};
    for i = 1:(length(input_string) - shingle_size + 1)
        shingle = input_string(i:i+shingle_size-1);
        shingles{end+1} = shingle;
    end
end
```

Figura 11 – Função generate_shingles

1. **Inicialização do Vetor de Shingles:**
    - **shingles = {};**: Inicializa um vetor vazio para armazenar os shingles.

2. **Geração de Shingles:**
    - O loop **for i = 1:(length(input_string) - shingle_size + 1)** itera sobre a string de entrada, criando shingles de tamanho fixo.
    - **shingle = input_string(i:i+shingle_size-1);**: Extrai uma subsequência (shingle) de tamanho **shingle_size** da string de entrada, começando na posição **i** e indo até **i+shingle_size-1**.
    - **shingles{end+1} = shingle;**: Adiciona o shingle ao vetor de shingles.
3. **Conclusão da Função:**
    - A função retorna o vetor **shingles** contendo todos os shingles gerados a partir da string de entrada.

Opção 5

No contexto da aplicação de análise de dados de filmes, a opção 5 foi introduzida para permitir ao utilizador procurar filmes com base nos géneros selecionados. Ao escolher a opção 5 no menu, a aplicação executa uma série de operações específicas. A seguir, apresentamos a implementação do case 5 no menu e uma explicação passo a passo dessa secção do código.

```
case 5
    bool = true;
    while bool
        count = 0;
        genre_data = [];
        genres_unique = display_available_genres(movies, 2);
        user_input = input('Select one or more genres (separated by '','',''): ', 's');
        data = strsplit(user_input, ',');
        if ~isempty(data)
            for i = 1:length(data)
                if str2double(data{i})<=0 || str2double(data{i})>length(genres_unique)
                    fprintf('ERROR! Wrong genre input on index %d\n', i);
                    count = count + 1;
                else
                    genre_data = [genre_data , cellstr(genres_unique{str2double(data{i})})];
                end
            end
            if count == 0
                bool = false;
            end
        end
    end
    search_similar_movies(movies, genre_data)
```

Figura 12 – case 5

1. **Loop de Validação de Géneros:**

    - **bool = true;**: Inicializa uma variável booleana para controlar o loop de validação.
    - **while bool**: Inicia um loop enquanto a variável booleana for verdadeira.
    - **count = 0;**: Inicializa uma contagem de erros.
    - **genre_data = [];**: Inicializa um vetor para armazenar os géneros selecionados.
    - **genres_unique = display_available_genres(movies, 2);**: Chama a função **display_available_genres** para mostrar os géneros disponíveis e obter a lista de géneros únicos.
    - **user_input = input('Select one or more genres (separated by '','',''): ', 's');**: Solicita ao utilizador selecionar um ou mais géneros, separados por vírgula.

- **data = strsplit(user_input, ',');**: Divide a entrada do utilizador em uma matriz de strings usando vírgulas como delimitadores.
- **if ~isempty(data)**: Verifica se o utilizador forneceu algum género.
    - **for i = 1:length(data)**: Loop para processar cada género fornecido pelo utilizador.
        - **str2double(data{i})<=0 || str2double(data{i})>length(genres_unique)**: Verifica se o índice do género fornecido está dentro dos limites válidos.
            - **fprintf('ERROR! Wrong genre input on index %d\n', i);**: Exibe uma mensagem de erro se o índice do género for inválido.
            - **count = count + 1;**: Incrementa a contagem de erros.
        - **genre_data = [genre_data , cellstr(genres_unique{str2double(data{i})})];**: Adiciona o género correspondente ao vetor **genre_data**.
    - **if count == 0**: Verifica se não houve erros durante a seleção de géneros.
        - **bool = false;**: Define a variável booleana como falsa para sair do loop.

2. **Chamada da Função search_similar_movies:**

    - **search_similar_movies(movies, genre_data)**: Chama a função **search_similar_movies** passando os dados dos filmes e os géneros selecionados pelo utilizador.

A função **search_similar_movies** é responsável por calcular a similaridade de Jaccard entre os géneros dos filmes e os géneros selecionados pelo utilizador. Vamos analisar cada parte desta função:

```
function search_similar_movies(movies,input_genres)

    % Calcular a similaridade de Jaccard e ordenar os filmes por similaridade de Jaccard e ano
    jaccard_indices = zeros(length(movies), 1);
    for i = 1:length(movies)
        % Acessar a célula de gêneros correspondente ao filme
        genres_cell = strsplit(movies{i, 3}, ',');

        % Calcular a similaridade de Jaccard
        intersection = numel(intersect(input_genres, genres_cell));
        unionSet = numel(union(input_genres, genres_cell));
        jaccard_indices(i) = intersection / unionSet;
        progress = (i*100)/length(movies);
        fprintf('Loading Results...%.2f %%\n', progress);
    end

    % Ordenar os filmes por similaridade de Jaccard e ano
    [~, indices] = sortrows([jaccard_indices, cell2mat(movies(:, 2))], [-1, -2]);

    % Mostrar os top 5 filmes
    fprintf('Top 5 Movies Similar to Genres "%s":\n', strjoin(input_genres, ', '));
    count = 0;
    for i = 1:length(indices)
        movie_index = indices(i);
        % Verificar se o índice de Jaccard é maior que zero (há alguma semelhança)
        if jaccard_indices(movie_index) > 0
            count = count + 1;
            fprintf('Movie: %-20s \tYear: %d\t\t Jaccard Index: %f\n', movies{movie_index, 1}, movies{movie_index, 2}, jaccard_indices(movie_index));
        end
        if count == 5
            fprintf('\n')
            break; % Parar após encontrar os top 5 filmes correspondentes
        end
    end
end
```

Figura 13 – Função search_similar_movies

1. **Cálculo da Similaridade de Jaccard:**

    - **jaccard_indices = zeros(length(movies), 1);**: Inicializa um vetor de zeros para armazenar os índices de Jaccard.
    - Loop para cada filme nos dados:
        - **genres_cell = strsplit(movies{i, 3}, ',');**: Divide os géneros do filme em uma célula de strings.
        - Loop para calcular a similaridade de Jaccard entre os géneros do filme e os géneros selecionados pelo utilizador.
            - **intersection = numel(intersect(input_genres, genres_cell));**: Calcula a interseção entre os géneros selecionados e os géneros do filme.
            - **unionSet = numel(union(input_genres, genres_cell));**: Calcula a união entre os géneros selecionados e os géneros do filme.
            - **jaccard_indices(i) = intersection / unionSet;**: Calcula e armazena o índice de Jaccard para o filme atual.

- **progress = (i*100)/length(movies); fprintf('Loading Results...%.2f %%\n', progress);**: Exibe o progresso do processo de carregamento.

2. **Ordenação dos Filmes por Similaridade de Jaccard e Ano:**

   - **[~, indices] = sortrows([jaccard_indices, cell2mat(movies(:, 2))], [-1, -2]);**: Ordena os índices dos filmes com base na similaridade de Jaccard decrescente e, em caso de empate, pelo ano do filme decrescente.

3. **Apresentação dos Top 5 Filmes:**
   - Loop para apresentar os top 5 filmes com base na similaridade de Jaccard.
     - **fprintf('Top 5 Movies Similar to Genres "%s":\n', strjoin(input_genres, ', '));**: Exibe o cabeçalho com os géneros selecionados.
     - **fprintf('Movie: %-20s \tYear: %d\t\t Jaccard Index: %f\n', movies{movie_index, 1}, movies{movie_index, 2}, jaccard_indices(movie_index));**: Exibe informações sobre cada filme, incluindo nome, ano e índice de Jaccard.
4. **Saída da Função:**
   - **fprintf('\n');**: Adiciona uma linha em branco para melhor formatação.

Conclusão

Com este trabalho, solidificámos os nossos conhecimentos na unidade curricular Métodos Probabilísticos para Engenharia Informática.

Para além disto, aprofundámos também os nossos conhecimentos de MATLAB, já que ficámos a conhecer novas funções que o mesmo disponibiliza, assim como consolidámos a implementação das nossas próprias funções, numa abordagem de clarificar o código por nós implementado.

Enfrentamos desafios específicos nas opções 4 e 5. Na opção 4, tivemos dificuldades na utilização do método minHash, e os resultados dos géneros não foram conforme o esperado. Na opção 5, observamos que os valores de índice de Jaccard não estavam a ser calculados corretamente.

Em síntese, este trabalho solidificou o nosso conhecimento de métodos probabilísticos e aprimorou as nossas habilidades no uso do MATLAB.