PL 4 - Algoritmos Probabilísticos

Trabalho de Métodos Probabilísticos para Engenharia Informática
Professores: Carlos Bastos, Amaro Sousa

José Diogo Cerqueira nº76758, João Monteiro nº114547, P5

Abstract

O objetivo deste trabalho consistiu no desenvolvimento de uma aplicação para pesquisa de filmes, que se baseasse na matéria lecionada no fim do semestre. Em especial, o desenvolvimento da aplicação focava-se na implementação de um método **minHash**, para cálculo da **similaridade de Jaccard** e, ainda, o desenvolvimento de um **filtro de Bloom** com contagem.

Keywords: *Filtros de Bloom, minHash, Jaccard, shingles.*

---

1. Desenvolvimento da aplicação

1.1. Opção 1

Inicialmente, foi necessário aceder aos dados que nos eram fornecidos, através do comando:

```
movies = readcell('films.txt', 'Delimiter', ',');
```

Listando todos os géneros que existiam neste novo cell array criado (**movies**), o objetivo era que o utilizador escolhesse um dos géneros de filmes existentes, para basear o restante das operações. Para tal armazenámos na variável **genres**, os vários géneros de filmes que apareciam no ficheiro. Com a ajuda da função Matlab *input*, atribuímos à variável **genreInput** aquilo que o utilizador escrevesse na linha de comandos e, por fim, verificámos se essa variável correspondia a um dos géneros que tinhamos disponível.

```
numMovies = height(movies);

genres = unique(movies(:,3));

fprintf(['Genres:\t%-16s\t%-8s\t%-10s\t%-10s\n\t\t%-16s\t%-8s\t%-10s\t%-10s' ...
        '\n\t\t%-16s\t%-8s\t%-10s\t%-10s\n\t\t%-16s\t%-8s\t%-10s\t%-10s\n\t' ...
        '\t%-16s\t%-8s\t%-10s\t%-10s\n'], genres{1:20});

genreInput = input("Select a genre: ", "s");
while true
    upperLetter = upper(genreInput(1));
    genreUpper = [upperLetter, genreInput(2:end)];
    if ismember(genreUpper, genres)
        break;
    end
    fprintf('ERROR inputing the genre!\n');
    genreInput = input("Select a genre: ", "s");
end
```

De seguida, podemos ver uma implementação do menu que era pedido, para ser apresentado ao utilizador:

```
while true
    disp('-------------------------------------');
    fprintf('SELECTED GENRE: %s\n', genreUpper);
    disp('-------------------------------------');
    fprintf('1 - Change selected Genre\n');
    fprintf('2 - No. of movies of selected Genre on given years\n');
```

```
    fprintf('3 - Search movie titles of selected Genre\n');
    fprintf('4 - Search movies based on Genres\n');
    fprintf('5 - Exit\n');
    disp('------------------------------------');
    option = input('Select an option: ');
    switch(option)
```

Basicamente, na opção 1 o processo era exatamente igual ao que era pedido inicialmente ao utilizador. Porém, neste caso específico, o objetivo era alterar o género previamente escolhido.

```
        case 1
            disp('------------------------------------');
            fprintf(['Genres:\t%-16s\t%-8s\t%-10s\t%-10s\n\t\t%-16s\t%-8s\t%-10s\t%-10s'
    ...
                    '\n\t\t%-16s\t%-8s\t%-10s\t%-10s\n\t\t%-16s\t%-8s\t%-10s\t%-10s\n\t'
    ...
                    '\t%-16s\t%-8s\t%-10s\t%-10s\n'], genres{1:20});
            disp('------------------------------------');
            genreInput = input("Select a genre: ", "s");
            while true
                upperLetter = upper(genreInput(1));
                genreUpper = [upperLetter, genreInput(2:end)];
                if ismember(genreUpper, genres)
                    break;
                end
                fprintf('ERROR inputing the genre!\n');
                genreInput = input("Select a genre: ", "s");
            end
            continue;
```

Segue uma pequena demonstração desta parte do script:

```
Genres: (no genres listed)  Action    Adventure   Animation
    Children          Comedy    Crime       Documentary
    Drama             Fantasy   Film-Noir   Horror
    IMAX              Musical   Mystery     Romance
    Sci-Fi            Thriller  War         Western
Select a genre: action
------------------------------------
SELECTED GENRE: Action
------------------------------------
1 - Change selected Genre
2 - No. of movies of selected Genre on given years
3 - Search movie titles of selected Genre
4 - Search movies based on Genres
5 - Exit
------------------------------------
Select an option: 1
------------------------------------
Genres: (no genres listed)  Action    Adventure   Animation
    Children          Comedy    Crime       Documentary
    Drama             Fantasy   Film-Noir   Horror
    IMAX              Musical   Mystery     Romance
    Sci-Fi            Thriller  War         Western
------------------------------------
Select a genre: western
------------------------------------
SELECTED GENRE: Western
------------------------------------
1 - Change selected Genre
2 - No. of movies of selected Genre on given years
3 - Search movie titles of selected Genre
4 - Search movies based on Genres
5 - Exit
------------------------------------
Select an option: 5
Exiting the program
```

1.2. Opção 2

O **Bloom Filter com Contagem** foi implementado com base no algoritmo do **Bloom Filter** normal que desenvolvemos previamente, tendo apenas de alterar a função de inserção e de verificação, para que em vez de se inserir/verificar um bit, ser apenas necessário inserir/verificar um número inteiro. Então,

em vez de inicializar o **Bloom Filter** com falsos, inicializámos com zeros, e em vez de incrementar o bit, incrementámos o número inteiro. Da mesma forma, em vez de verificar se o bit é 1, verificámos se o número inteiro é maior que 0.

Para inserir um elemento no **Bloom Filter**, temos de aplicar as k funções de dispersão ao elemento, e incrementar o i-ésimo elemento do **Bloom Filter** para cada função de dispersão, onde i é o resultado da função de dispersão. Para verificar se um elemento pertence ao **Bloom Filter**, fazemos o mesmo processo, mas em vez de incrementar, verificamos qual o valor do i-ésimo elemento.

```
function CBF = runCountingBloomFilter(movies)
    tic;
    fprintf("\n==========================================================================")

    numMovies = height(movies);                          % Number of movies
    genresPerMovie = 3;                                  % Number of genres per movie
    p = 0.01;                                            % Pretended probability of false
    positives
    numElem = numMovies * genresPerMovie;                % Number of elements to insert in
    the counting bloom filter
    n = int32(-log(p) * numElem / (log(2)^2));           % Size of ideal bloom filter
    k = int32(n * log(2) / numElem);                     % Number of hash functions

    % 1. Criar um Bloom Filter com n posicoes e k funcoes de dispersao
    CBF = CountingBloomFilter(n, k);

    fprintf('\n1. Bloom Filter criado com sucesso, com %d funcoes de dispersao\n', k);

    % 2. Inserir todos pares (ano, genero) no Bloom Filter
    wb = waitbar(0, 'Inserindo elementos no Bloom Filter...');
    for i = 1:numMovies
        % Update waitbar every 2500 movies for performance reasons
        if mod(i, 2500) == 0
            waitbar(i/numMovies, wb);
        end
        CBF = CountingBloomFilterInsert(CBF, [num2str(movies{i,2}) movies{i,3}]);
    end
    close(wb);
    fprintf('\n2. Todos os pares (ano, genero) inseridos no Bloom Filter\n');

    % 3. Verificar se todos os pares (ano, genero) pertencem ao Bloom Filter
    wb = waitbar(0, 'Verificando elementos no Bloom Filter...');
    for i = 1:numMovies
        % Update waitbar every 2500 movies for performance reasons
        if mod(i, 2500) == 0
            waitbar(i/numMovies, wb);
        end
        check = CountingBloomFilterCheck(CBF, [num2str(movies{i,2}) movies{i,3}]);
        if ~check
            fprintf('\n3. Erro: O par (ano, genero) (%d, %s) nao pertence ao Bloom Filter\n
    ', movies{i,2}, movies{i,3});
            break;
        end
    end
    close(wb);
    fprintf('\n3. Todos os pares (ano, genero) verificados no Bloom Filter\n');
    disp(toc);
end

function CBF = CountingBloomFilter(n, k)
    CBF.n = n;              % Tamanho do Bloom Filter
    CBF.k = k;              % Numero de funcoes de dispersao
    CBF.cbf = zeros(1, n);   % Inicializar o Bloom Filter
    CBF = CountingBloomFilterInitHF(CBF); % Inicializar as funcoes de dispersao
end

function CBF = CountingBloomFilterInitHF(CBF)
    hashFunctions = {@(element) string2hash(element, 'sdbm'), ...
                     @(element) string2hash(element, 'djb2'), ...
                     @(element) DJB31MA(element, 5381)};

    CBF.hashFunctions = cell(1, CBF.k);
    for i = 1:CBF.k
        CBF.hashFunctions{mod(i,3)+1} = @(element) mod(hashFunctions{mod(i,3)+1}(element),
    CBF.n) + 1;
```

```
    end
end

function CBF = CountingBloomFilterInsert(CBF, x)
    xCell = cell(1, CBF.k);     % Inicializar o array de elementos a inserir
    for i = 1:CBF.k
        xCell{i} = [x num2str(i)];
        index = CBF.hashFunctions{mod(i,3)+1}(xCell{i});
        CBF.cbf(index) = CBF.cbf(index) + 1;
    end
end

function num = CountingBloomFilterCheck(CBF, x)
    xCell = cell(1, CBF.k);         % Inicializar o array de elementos a inserir
    for i = 1:CBF.k
        xCell{i} = [x num2str(i)];  % Tem de ser o mesmo que foi usado para inserir
        % Verificar o i-esimo elemento do Bloom Filter
        index = CBF.hashFunctions{mod(i,3)+1}(xCell{i});
        if CBF.cbf(index) == 0
            num = 0;
            return;
        end
        num = CBF.cbf(index);
    end
end
```

- **CountingBloomFilter**: - Cria um Counting Bloom Filter com n posições e k funções de dispersão; - O valor de n é calculado com base na fórmula para o tamanho ideal do Bloom Filter: -n * ln(p) / $(\ln(2)^2)$;

- O valor de k é calculado com base na fórmula: k = n * ln(2) / m, onde n é o número de elementos a inserir e m é o tamanho do Bloom Filter.

- **CountingBloomFilterInitHF**: - Inicializa as funções de dispersão; - As funções de dispersão são inicializadas com base em 3 funções de dispersão; - Estas 3 funções de dispersão são aplicadas a cada elemento a inserir sempre com um salt diferente, para evitar colisões e sequencialmente na mesma ordem.

- **CountingBloomFilterInsert**: - Insere um elemento no Bloom Filter; - Para inserir um elemento no Bloom Filter, temos de aplicar as k funções de dispersão ao elemento, e incrementar o i-ésimo elemento do Bloom Filter para cada função de dispersão, onde i é o resultado da função de dispersão; - É utilizado um salto diferente para cada função de dispersão, para evitar colisões e sempre com a mesma sequência de funções de dispersão.

- **CountingBloomFilterCheck**: - Verifica qual o número de vezes que um elemento x foi inserido no Bloom Filter; - Para verificar se um elemento pertence ao Bloom Filter, fazemos o mesmocprocesso que na inserção, mas em vez de incrementar, verificamos qual o valor do i-ésimo elemento.

1.3. Opção 3

Para a terceira opção, era pedido que se fizesse uma procura de filmes de um dado género, baseado no título desse filme. Para tal, o objetivo era calcular a similaridade de **Jaccard** entre strings (a procura e o título dos diversos filmes). Para tal, era necessária a implementação de um método **minHash**.

Antes de tudo, procedemos ao processamento da string que o utilizador viria a introduzir. Também criámos um cell array **moviesbyGenre**, onde apenas tínhamos os filmes do género que tinha sido escolhido previamente.

```
        case 3
            movieSearch = input('Insert a string: ', 's');
            k = 1;
            for i = 1:numMovies
                for j = 3:12
                    if strcmp(movies{i, j}, genreUpper)
                        for idx = 1:12
                            moviesbyGenre{k, idx} = movies{i, idx};
                        end
                        k = k + 1;
                    end
                end
            end
```

Desenvolvemos quatro funções: - A primeira para a criação dos *shingles*; - A segunda, para comparar as assinaturas, ou seja, para calcular a similaridade de Jaccard entre dois conjuntos; - A terceira função é uma função **minHash**; - Por último, uma função para obter as assinaturas e guardá-las num array;

```
function shingles = createShingles(str, n)
    shingles = cell(1, length(str) - n + 1);
    for i = 1:length(str) - n + 1
        shingles{i} = str(i:i+n-1);
    end
end

function jaccardSimilarity = compareMinHashSignatures(sig1, sig2)
    intersection = sum(sig1 == sig2);
    union = numel(unique([sig1, sig2]));
    jaccardSimilarity = intersection / union;
end

function minHash = minhash_DJB31MA(chave, seed, k)
    if nargin < 2
        seed = 127;
        k = 100;
    elseif nargin < 3
        k = 100;
    end

    len = length(chave);
    chave = double(chave);

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

    minHash = zeros(1, k);

    for j = 1:k
        h = mod(31 * h + j, 2^32 - 1);
        minHash(j) = h;
    end
end

function signatures = getSignatures(movies, k)
    titlesShingles = cell(length(movies), 1);
    signatures = inf(length(movies), k);
    wb = waitbar(0, 'Calculating minhash signatures...');
    for i = 1:length(movies)
        if mod(i, 10) == 0
            waitbar(i/length(movies), wb, 'Calculating minhash signatures...');
        end
        titlesShingles{i} = createShingles(movies{i,1}, 3);         % Obter os shingles do
    titulo do filme
        for j = 1:length(titlesShingles{i})                         % Para cada shingle do
    nome do filme
            key = titlesShingles{i}{j};                 % Obter o shingle
            minHash = minhash_DJB31MA(key, 127, 1000);              % Calcular a assinatura
    minhash
            signatures(i, :) = min(signatures(i, :), minHash);  % Guardar a assinatura
    minhash mais pequena
        end
    end
    close(wb);
end
```

Para o cálculo das similaridades, obtivemos, inicialmente, uma matriz de assinaturas minHash para cada filme no cell array **moviesbyGenre**. Procedemos também, de igual modo, ao cálculo do valor das assinaturas para a string que tinha sido introduzida pelo utilizador, armazenada em **movieSearch**. Por fim tendo o valor das assinaturas para ambos os casos, criámos uma matriz **similarities**, para armazenar os valores das similaridades entre a string introduzida e o nome de cada filme.

```
            numMoviesGenre = height(moviesbyGenre);
            signatures = getSignatures(moviesbyGenre, 1000);
            minHashSearch = inf(1, 1000);
            shingle = createShingles(movieSearch, 3);
```

```
            for j = 1:length(shingle)                          % Para cada shingle do nome do filme
                key = shingle{j};                  % Obter o shingle
                minHash = minhash_DJB31MA(key, 127, 1000);             % Calcular a assinatura minhash
                minHashSearch(1, :) = min(minHashSearch(1, :), minHash);  % Guardar a assinatura minhash mais pequena
            end
            similarities = zeros(1,numMoviesGenre);
            for i = 1:numMoviesGenre
                similarities(i) = (sum(minHashSearch(1, :) == signatures(i,:)) / 1000);
            end
```

Por fim, ordenamos a matriz, do maior valor para o menor e imprimimos no terminal a informação relativa a esse filme: similaridade de Jaccard, nome do filme e os géneros do respetivo filme.

```
            sortedSimilarities = sort(similarities, 'descend');
            ind = zeros(1, 5);
            for j = 1:5
                for k = 1:numMoviesGenre
                    if (sortedSimilarities(j) == distances(k))
                        if ~ismember(k, ind)
                            ind(1, j) = k;
                        end
                    end
                end
            end
            for j = 1:5
                printInfo(moviesbyGenre, ind(1, j), sortedSimilarities(j));
            end
```

Demonstramos de seguida uma implementação desta opção:

```
Genres: (no genres listed)  Action    Adventure   Animation
    Children          Comedy    Crime       Documentary
    Drama             Fantasy   Film-Noir   Horror
    IMAX              Musical   Mystery     Romance
    Sci-Fi            Thriller  War         Western
Select a genre: animation
-------------------------------------
SELECTED GENRE: Animation
-------------------------------------
1 - Change selected Genre
2 - No. of movies of selected Genre on given years
3 - Search movie titles of selected Genre
4 - Search movies based on Genres
5 - Exit
-------------------------------------
Select an option: 3
Insert a string: Toy Story
{1.0000} Toy Story - Adventure,Animation,Children,Comedy,Fantasy
{0.7790} Toy Story 3 - Adventure,Animation,Children,Comedy,Fantasy,IMAX
{0.7780} Toy Story 2 - Adventure,Animation,Children,Comedy,Fantasy
{0.7730} Toy Story 4 - Adventure,Animation,Children,Comedy
{0.3920} Toy Story of Terror - Animation,Children,Comedy
-------------------------------------
SELECTED GENRE: Animation
-------------------------------------
1 - Change selected Genre
2 - No. of movies of selected Genre on given years
3 - Search movie titles of selected Genre
4 - Search movies based on Genres
5 - Exit
-------------------------------------
Select an option: 5
Exiting the program
```

1.4. Opção 4

Na quarta opção era expectável que o utilizador inseri-se uma série de géneros de filmes (ou nenhum) separados por vírgulas. O objetivo era apresentar os filmes cujo conjunto de géneros fosse mais similar ao conjunto composto pela lista introduzida. Para tal, primeiro processámos o input do utilizador. Caso não

introduzisse nada, a lista de géneros iria ser constituída, apenas, pelo género atual. Caso contrário essa lista ia ser aumentada consoante os géneros introduzidos:

```
            %  Ask for the number of genres to search, it can be none
            userInput = input('Select additional Genres separated by (',') (press ENTER when no desired additional Genres):  ', 's');
            if isempty(userInput)
                movieGenres = cell(1, 1);
                movieGenres{1} = genreInput;
            else
                % Normalize the input, making the first letter uppercase and the rest lowercase
                upperLetter = upper(userInput(1));
                userInput = [upperLetter, userInput(2:end)];
                % Get the genres from the input
                movieGenres = split(userInput, ',');
                % Check if the input genres are valid
                for i = 1:length(movieGenres)
                    if ~ismember(movieGenres{i}, genres)
                        fprintf('ERROR inputing the genre!\n');
                        movieGenres = genreInput;
                        break;
                    end
                end
                % Add genreInput to the movieGenres and remove duplicates
                movieGenres{end+1} = genreInput;
                movieGenres = unique(movieGenres);

            end
```

De seguida, criámos as assinaturas para os géneros selecionados. Através da função **minhashDJB31MA**, previamente demonstrada, demos o valor à variável minHash para, de seguida, criar a matriz de assinaturas para os géneros selecionados. Noutro ficheiro, criámos a matriz de assinaturas para todos os filmes. De igual modo, cálculamos o valor da similaridade entre cada filme e a lista de géneros e atribuímos tudo a uma matriz **similarities**.

```
            % Create signatures for the selected genres
            selectedSign = inf(1, kMinHash);
            for i = 1:length(movieGenres)                    % For each genre
                key = char(movieGenres(i));                  % Get the genre
                minHash = minhash_DJB31MA(key);              % Create the minhash signature
                selectedSign = min(selectedSign, minHash);  % Get the minimum value for each hash function
            end

            % Compare the signatures of the movies with the signatures of the selected genres
            similarity = zeros(numMovies, 1);
            for i = 1:numMovies
                similarity(i) = compareMinHashSignatures(selectedSign, signaturesGenres(i,:));
            end

            %% Outro Ficheiro
            genres = movies(1, 3:10);                                % Obter os generos
            k = 100;                                                 % Numero de funcoes de hash a usar
            signaturesGenres = inf(length(movies), k);               % Guardar as assinaturas minhash de cada filme
            moviesGenres = cell(length(movies), 1);                  % Guardar os generos de cada filme
            wb = waitbar(0, 'Calculating minhash signatures...');
            for i = 1:length(movies)                                 % Para cada filme
                if mod(i, 10) == 0
                    waitbar(i/length(movies), wb, 'Calculating minhash signatures...');
                end
                moviesGenres{i} = movies(i, 3:10);                                  % Obter os generos do filme
                for j = 1:length(moviesGenres{i})                                   % Para cada genero do filme
                    key = moviesGenres{i}{j};                                       % Obter o genero
                    if ~isNotMissing(key)                                           % Se o genero for missing
```

```
                        continue;                                                   % Passar ao proximo genero
                    end
                    minHash = minhash_DJB31MA(key);                                 % Calcular a assinatura minhash
                    signaturesGenres(i, :) = min(signaturesGenres(i, :), minHash);  % Guardar a assinatura minhash mais pequena
                end
            end
            close(wb);

            save('signaturesGenres.mat', 'signaturesGenres', 'moviesGenres', 'genres');
```

Por último ordenamos a matriz **similarities**, de modo a apresentarmos os 5 filmes com maior similaridade:

```
            % Sort the movies by similarity, then most recent year
            [~, idx] = sortrows([similarity, cell2mat(movies(:, 2))], [-1, -2]);

            % Select the top 5 movies
            idx = idx(1:5);

            % Print the top 5 movies
            fprintf('\n%-s\t%-40s\t%-16s\t\n', 'Year', 'Movie', 'Similarity');
            for i = 1:length(idx)
                % Print the name, year and similarity of the movie
                fprintf('%-d\t%-40s\t%-.3f\t\n', movies{idx(i), 2}, movies{idx(i), 1}, similarity(idx(i)));
            end
```

Apresentamos um breve exemplo da execução deste bloco de código:

```
Genres: (no genres listed)  Action    Adventure   Animation
    Children          Comedy    Crime       Documentary
    Drama             Fantasy   Film-Noir   Horror
    IMAX              Musical   Mystery     Romance
    Sci-Fi            Thriller  War         Western
Select a genre: action
-------------------------------------
SELECTED GENRE: Action
-------------------------------------
1 - Change selected Genre
2 - No. of movies of selected Genre on given years
3 - Search movie titles of selected Genre
4 - Search movies based on Genres
5 - Exit
-------------------------------------
Select an option: 4
Select additional Genres separated by (',') (press ENTER when no desired additional Genres)
    :  Comedy

Year  Movie                                   Similarity
2019  Celebration                             0.282
2019  The Farewell                            0.282
2019  Sebastian Maniscalco: Stay Hungry       0.282
2019  Non ci resta che il crimine             0.282
2019  The Beach Bum                           0.282
-------------------------------------
SELECTED GENRE: Action
-------------------------------------
1 - Change selected Genre
2 - No. of movies of selected Genre on given years
3 - Search movie titles of selected Genre
4 - Search movies based on Genres
5 - Exit
-------------------------------------
Select an option: 5
Exiting the program
```

2. Fundamentação das opções tomadas na implementação

2.1. Tamanho dos Shingles

Para o tamanho dos shingles, nós consideramos que 3, seria o tamanho mais adequado, uma vez que nos pareceu o número mais equilibrado. Com 2 podia haver muitas semelhanças, por ser uma amostra pequena, e seria provável a obtenção de valores deturpados. Para tamanhos de shingles maiores, também o valor sairia deturpado, uma vez que as strings tinham de ser muito parecidas para dar *match* em 4 ou até 5 carateres seguidos. Daí a nossa escolha pareceu-nos a mais adequada.

2.2. Número de funções de dispersão

No desenvolvimento, ao usar 100 funções de dispersão, percebemos que o nosso código devolvia os valores expectáveis e não era muito demorado, portanto optamos por esse número.

2.3. Dimensionamento dos Filtros de Bloom

```
1    n = int32(-log(p) * numElem / (log(2)^2));          % Size of ideal bloom filter
2    k = int32(n * log(2) / numElem);                    % Number of hash functions
```

No nosso código foram implementadas estas duas fórmulas que representam o tamanho ideal para o Filtro de Bloom e também o número para as funções de dispersão.

3. Conclusões

Este trabalho teve como objetivo, essencialmente, a criação de uma aplicação aplicando os conhecimentos obtidos relativamente a algoritmos probabilísticos. Este trabalho, permitiu-nos, acima de tudo, a um desenvolvimento de capacidades e competências em Matlab e, especialmente, a operar com **Filtros de Bloom** e com o método **minHash**.