



Métodos Probabilísticos para Engenharia Informática

Guião PL04

Ano letivo 2023/2024

Turma P4

Ricardo Quintaneiro
NMec: 110056

Rúben Garrido
NMec: 107927

21 de dezembro de 2023

Índice

1	Introdução	2
2	Material fornecido	2
3	Desenvolvimento	3
3.1	Hash function	3
3.2	Opção 1	3
3.3	Opção 2	3
3.3.1	Criação do filtro de Bloom	3
3.3.2	Inserção de elementos	4
3.3.3	Pesquisa	4
3.4	Opção 3	5
3.4.1	Criação do filtro de Bloom	5
3.4.2	Inserção de elementos	5
3.4.3	Pesquisa	6
3.5	Opção 4	7
3.5.1	Estrutura de dados	7
3.5.2	Assinaturas	7
3.5.3	Pesquisa	8
3.6	Opção 5	9
3.6.1	Estrutura de dados	9
3.6.2	Assinaturas	9
3.6.3	Pesquisa	9

1 Introdução

No âmbito da Unidade Curricular de Métodos Probabilísticos para Engenharia Informática, foi-nos proposto realizar um trabalho prático relativo a Hash Functions, Bloom Filter e MinHash. Este consiste em desenvolver um script de consulta de filmes e géneros, que se relacionam entre si.

Este relatório serve para documentar os processos de programação dos scripts e para fundamentar as escolhas das opções relativas à implementação dos métodos probabilísticos, como por exemplo, o número de funções de dispersão a usar, o tamanho de shingles e a dimensão dos filtros de Bloom.

2 Material fornecido

Foi-nos fornecido um ficheiro CSV (*Comma-Separated Values*), `movies.csv`, onde constam informações sobre mais de 50000 filmes.

Quanto à estrutura, na primeira coluna está presente o título, na segunda o ano e nas restantes os vários géneros associados ao filme.

3 Desenvolvimento

3.1 Hash function

Recorremos à função muxDJB31MA para efetuar o *hashing* em todas as ocasiões onde este é necessário.

Esta é uma modificação da função DJB31MA utilizada nas aulas, onde é usado um argumento *k* correspondente ao número de funções de *hash*. Isto permite reduzir a complexidade computacional, já que deixa de ser necessário invocar a função e percorrer a string para cada função de *hash*.

```
1 function aux = muxDJB31MA(chave, seed, k)
2 len = length(chave);
3 chave = double(chave);
4 h = seed;
5 aux = zeros(1, k);
6 for i=1:len
7     h = mod(31 * h + chave(i), 2^32 -1) ;
8 end
9 for i = 1:k
10    h = mod(31 * h + i, 2^32 -1) ;
11    aux(i) = h;
12 end
13 end
```

3.2 Opção 1

Filtrámos, através de um ciclo *for*, os demais géneros, de modo a obter um vetor de géneros únicos, ignorando os elementos do *cell array* que estão *missing*.

```
1 genres = strings([]);
2 [x, y] = size(movies);
3
4 for i = 1:x
5     for j = 3:y
6         if ~ismissing(movies{i, j})
7             g = movies{i, j};
8             if ~strcmp(g, '(no genres listed)') && ~ismember(
                convertCharsToStrings(movies{i, j}), genres)
9                 genres = [genres convertCharsToStrings(movies{i, j}
10                    )];
11             end
12         end
13     end
14 end
```

3.3 Opção 2

3.3.1 Criação do filtro de Bloom

Para inicializar um filtro de bloom, recorre-se a um vetor com o tamanho passo como argumento da função `bloomFilterInitialization`.

Recorremos à fórmula $\frac{-n \times \log(n)}{\log(2)^2}$ para obter o tamanho adequado do filtro, onde n é a probabilidade de falsos positivos desejada. Escolhemos n como 0.001 porque consideramos ser um equilíbrio entre uma percentagem baixa de erro e o consumo de memória do filtro.

```

1 m = ceil(-n*log(0.001)/(log(2))^2);
2 genre_bloom = bloomFilterInitialization(m);
3
4 function bloom = bloomFilterInitialization(n)
5 bloom = zeros(1, n, 'uint16');
6 end

```

3.3.2 Inserção de elementos

Para inserir elementos, utilizamos a função `bloomFilterInsert`, onde se obtém o conjunto de *hashes* com base na *key* inserida, e, ao valor, se calcula o resto da divisão pelo tamanho do filtro de Bloom. Uma vez que este filtro envolve contagem, é feita a soma de 1 ao invés de definir como *true*.

Utilizamos esta função para inserir todos os géneros que se encontram no cell array criado pelo ficheiro CSV, ignorando os valores *missing*.

```

1 n = length(genres);
2 m = ceil(-n*log(0.001)/(log(2))^2);
3 k_g = round((m/n)*log(2));
4
5 for i = 1:x
6     for j = 3:y
7         if ~ismissing(movies{i, j})
8             genre_bloom = bloomFilterInsert(genre_bloom, movies{i,
9                 j}, k_g);
10        end
11    end
12
13 function bloom = bloomFilterInsert(bloom, key, k)
14 m = length(bloom);
15 aux = muxDJB31MA(key, 127, k);
16 for i = 1:k
17     hash = mod(aux(i), m) + 1;
18     bloom(hash) = bloom(hash) + 1;
19 end
20 end

```

3.3.3 Pesquisa

Ao *char* array obtido através do *input*, é feita uma conversão para *string*, onde depois se invoca a função `bloomFilterCheck`.

Esta possui a mesma lógica da função `bloomFilterInsert` (ver secção 3.3.2), embora guarde os vários valores obtidos pela função de *hash* e retorne o menor.

Caso o valor obtido seja 0, significa que o género não existe.

```

1 genre = convertStringsToChars(input("Select a genre: ", "s"));
2 if ~ismember(genre, genres)

```

```

3     fprintf("Genre not found!\n")
4 else
5     count = bloomFilterCheck(genre_bloom, genre, k);
6     fprintf("%d movies with genre %s\n", count, genre);
7 end
8
9 function count = bloomFilterCheck(bloom, key, k)
10     m = length(bloom);
11     aux = muxDJB31MA(key, 127, k);
12     count = [];
13     for i = 1:k
14         key = [key num2str(i)];
15         hash = mod(aux(i), m) + 1;
16         count = [count bloom(hash)];
17     end
18
19     if ~isempty(count)
20         count = min(count);
21     else
22         count = 0;
23     end
24 end

```

O output desta opção é o seguinte:

```

1 Select a genre:
2   Comedy
3
4 16124 movies with genre Comedy

```

3.4 Opção 3

3.4.1 Criação do filtro de Bloom

Para a criação do filtro, recorreu-se à mesma estratégia da opção 4 (ver secção 3.3.1).

3.4.2 Inserção de elementos

Para a inserção dos pares (Género, Ano) no bloom, recorreu-se à mesma função da secção 3.3.2, a `bloomFilterInsert`.

No entanto, uma vez que esta função aceita apenas strings, houve a necessidade de concatenar o par. Assim, para cada par, unimos ambos os valores numa string de formato "GéneroAno", e inserimos num vetor `genres_years`.

```

1 totalElements = x * (y - 2);
2 genres_years = strings(1, totalElements);
3 count = 1;
4
5 for i = 1:x
6     for j = 3:y
7         if ~ismissing(movies{i, j})
8             genre_year = strcat(movies{i, j}, string(movies{i, 2}))
9             ;

```

```

9         genres_years(count) = genre_year;
10        count = count + 1;
11    end
12 end
13 end
14
15 genres_years = genres_years(1:count-1);
16
17 n_gy = length(unique(genres_years));
18 m_gy = ceil(-n_gy*log(0.0001)/(log(2))^2);
19 k_gy = round((m_gy/n_gy)*log(2));
20
21 for i = 1:length(genres_years)
22     genre_year_bloom = bloomFilterInsert(genre_year_bloom,
23     convertStringsToChars(genres_years(i)), k_gy);
24 end

```

3.4.3 Pesquisa

Em primeiro lugar, é obtido o gênero e o ano através da função `strsplit`, que separa o input recebido.

Caso algum destes não se encontre no formato pedido, ou caso o gênero não exista, a opção termina com uma mensagem de erro.

```

1 a = input("Select a genre and a year (separated by ','): ", "s");
2 a = strsplit(a, ',');
3
4 genre = a{1};
5 if ~ismember(genre, genres)
6     fprintf("Genre not found!\n")
7     continue
8 end
9
10 year = a{2};
11 if isempty(str2num(year))
12     fprintf("Invalid number!\n")
13     continue
14 end

```

Por fim, é aplicada a mesma estratégia da opção 2, utilizando a função `bloomFilterCheck`. No entanto, esta é chamada duas vezes, uma para o gênero e outra para o par (gênero, ano), de modo a poder cruzar informações no output.

```

1 genre_year = convertStringsToChars(strcat(genre, year));
2 count_genre = bloomFilterCheck(genre_bloom, genre, k_g);
3 count_genre_year = bloomFilterCheck(genre_year_bloom, genre_year,
4 k_gy);
5 fprintf("%d movies released in %s with genre %s\n",
6 count_genre_year, year, genre);

```

O output é o seguinte:

```

1 Select a genre and a year (separated by ','):

```

```
2      Comedy , 1995
3
4 163 movies released in 1995 with genre Comedy
```

3.5 Opção 4

3.5.1 Estrutura de dados

Para criar as assinaturas necessárias à execução desta opção, é necessário criar uma estrutura de dados adequada, que permita posteriormente efetuar a comparação entre a string introduzida e os vários títulos.

Assim, para cada filme, são criados shingles a partir do título. Estes têm comprimento 2, uma vez que é o número que permite obter uma maior fiabilidade nos resultados, pelo facto de existir uma maior granularidade.

```
1 [Set_title, Nm] = createMovieTitleStructure(movies);
2
3 function [Set, Nm] = createMovieTitleStructure(movies)
4 % For each movie, get its title split in shingles.
5 Nm = length(movies);
6 Set = cell(Nm,1);
7
8 for n = 1:Nm % Lines (movies)
9     title = movies{n,1};
10    for i = 1:length(title)-1
11        Set{n} = [Set{n} convertCharsToStrings(title(i:i+1))];
12    end
13 end
14 end
```

3.5.2 Assinaturas

As assinaturas são geradas com a função `getSignatures`. Para cada conjunto do set, esta função faz uso do algoritmo MinHash: calcula a *hash* para cada elemento e guarda o menor valor obtido.

Foram utilizadas 100 funções de hash (ou seja, $K = 100$), uma vez que consideramos ser um número adequado tendo em conta o tamanho do set em questão.

```
1 signatures_title = getSignatures(Set_title, Nm, 100);
2
3 function signatures = getSignatures(Set, Nm, K)
4 signatures = inf(Nm, K);
5
6 for n = 1:Nm
7     set_n = Set{n};
8     for i = 1:length(set_n)
9         key = num2str(set_n(i));
10        h_out = muxDJB31MA(key, 127, K);
11        signatures(n,:) = min(h_out, signatures(n,:));
12    end
13 end
14 end
```


3.5.3 Pesquisa

Para a string introduzida na consola do MATLAB, são repetidos os passos das secções 3.5.1 e 3.5.2, de modo a obter as suas assinaturas.

```
1 a = input("Insert a string: ", "s");
2 Set = cell(1,1);
3
4 for i = 1:length(a)-1
5     Set{1} = [Set{1} convertCharsToStrings(a(i:i+1))];
6 end
7
8 k = size(signatures_title, 2);
9 signatures = getSignatures(Set, 1, k);
```

De seguida, comparam-se ambas as matrizes de assinaturas para obter a distância (e consequentemente a similaridade) de Jaccard.

```
1 size_titles = size(signatures_title, 1);
2 similarities = zeros(1, size_titles);
3
4
5 for n = 1:size_titles
6     similarities(n) = 1 - (sum(signatures ~= signatures_title(n, :)) / k);
7 end
```

Após obter o vetor de similaridades, é efetuada neste uma ordenação decrescente quanto à similaridade de Jaccard e aplicado um limite de 5 filmes, de acordo como é pedido no enunciado.

```
1 [sortedSimilarities, indices] = sort(similarities, 'descend');
2 topSimilarities = sortedSimilarities(1:5);
3 topTitles = movies(indices(1:5));
4 topGenres = Set_genre(indices(1:5));
```

Por fim, e uma vez que temos um conjunto de shingles, efetuamos uma concatenação entre estes de modo a obter o título completo.

```
1 fprintf("Top 5 Similar Titles:\n");
2 concatenatedTitle = strings(length(topTitles{i}) + 1);
3 for i = 1:5
4     displayString = sprintf("\t%s - %f - Genres: ", topTitles{i},
5                             topSimilarities(i));
6     for j = 1:length(topGenres{i})
7         displayString = strcat(displayString, convertCharsToStrings
8                                 (topGenres{i}{j}), ", ");
9     end
10    fprintf('%s\n', displayString);
11 end
```

O output é, conforme esperado:

```
1 Insert a string:
2 Love
3
```

```

4 Top 5 Similar Titles:
5   Love - 1.000000
6   Lover - 0.690000
7   Love65 - 0.530000
8   Loveling - 0.520000
9   Lovelife - 0.520000

```

3.6 Opção 5

3.6.1 Estrutura de dados

A estrutura de dados é semelhante à encontrada para a opção 4 (ver secção 3.5.2). No entanto, ao invés de criar *shingles*, o set inclui os conjuntos de géneros para cada filme.

```

1 [Set_genre, Nm] = createMovieGenreStructure(movies);
2
3 function [Set, Nm] = createMovieGenreStructure(movies)
4 % For each movie, get its genres
5 Nm = length(movies);
6 Set = cell(Nm,1);
7
8 for n = 1:Nm % Lines (movies)
9     for g = 3:12 % Columns (genres)
10         if ~ismissing(movies{n, g})
11             Set{n} = [Set{n} convertCharsToStrings(movies{n,g})];
12         end
13     end
14 end
15 end

```

3.6.2 Assinaturas

O conjunto de assinaturas para esta opção foi obtido da mesma forma que a opção 4, utilizando a *hash function* (ver secção 3.5.2).

```

1 signatures_genre = getSignatures(Set_genre, Nm, 100);

```

3.6.3 Pesquisa

É pedida uma string de géneros no formato CSV pelo input do MATLAB, ao que é efetuado um split por vírgula, de modo a obter os diversos géneros que nesta se encontram. Com isto, é possível verificar se algum dos géneros introduzidos não pertence ao conjunto de géneros existentes na matriz.

```

1 a = input("Select one or more genres (separated by ','): ", "s");
2 a = strsplit(a, ',');
3
4 arevalid = true;
5 for i = 1:length(a)
6     if ~ismember(a{i}, genres)
7         fprintf("Genre %s is invalid!\n", a{i});
8         arevalid = false;
9         break

```

```

10     end
11 end
12 if ~arevalid
13     continue
14 end

```

De seguida, é criada uma matriz de assinaturas para os vários géneros, tal como na opção 4.

```

1 Set = strings(length(a),1);
2 for i= 1:length(a)
3     Set(i,1) = convertCharsToStrings(a(i));
4 end
5 b = cell(1,1);
6 b{1} = Set;
7 k = size(signatures_genre, 2);
8 signatures = getSignatures(b, length(b), k);

```

É efetuada uma comparação de assinaturas para obter a similaridade de Jaccard, seguido de uma ordenação e limite de output.

```

1 size_genre = size(signatures_genre, 1);
2 similarities = zeros(1, size_genre);
3
4 for n = 1:size_genre
5     similarities(n) = 1 - (sum(signatures ~= signatures_genre(n, :)
6         ) / k);
7 end
8 [sortedSimilarities, indices] = sort(similarities, 'descend');
9 topSimilarities = sortedSimilarities(1:5);
10 topTitles = movies(indices(1:5), 1);

```

Por fim, é realizado um print dos vetores, que contêm os demais títulos e similaridades.

```

1 fprintf("Top 5 Similar Titles:\n");
2 for i = 1:5
3     fprintf("\t%s - %f\n", topTitles{i}, topSimilarities(i));
4 end

```

O output é, como esperado:

```

1 Select one or more genres (separated by ','):
2     Comedy,Action,IMAX
3
4 Top 5 Similar Titles:
5     Night at the Museum: Battle of the Smithsonian - 1.000000
6     Men in Black III (M.III.B.) (M.I.B.) - 0.770000
7     I Love Trouble - 0.750000
8     Naked Gun 33 1/3: The Final Insult - 0.750000
9     Low Down Dirty Shame, A - 0.750000

```