Métodos Probabilísticos para Engenharia Informática
2023/2024

PL 4

Algoritmos Probabilísticos

Alunos da Turma P4:

João Varela 113780

Carolina Prata 114246

Introdução

No âmbito da disciplina de Métodos Probabilísticos para Engenharia Informática, foi proposto como trabalho prático de avaliação do guião PL 04 – Algoritmos Probabilísticos, o desenvolvimento de uma aplicação, em MATLAB, com algumas funcionalidades de um sistema de dados sobre filmes.

Foi nos fornecidos um ficheiro chamado **movies.csv** com informações sobre as quais foi analisada cada situação possível.

Utilizamos técnicas de processamento de dados, como filtros **Bloom** e **MinHash**, para eficientemente gerenciar e analisar um conjunto de filmes. O objetivo é fornecer dados sobre a distribuição de géneros de filmes, a frequência de filmes por ano e género, e encontrar títulos de filmes semelhantes com base em palavras-chave.

Script 1

O ponto de partida deste guião é o script Script1.m que vai ler e processar todos os dados necessários para que o segundo script (Script2.m) funcione sem serem necessários cálculos constantes e o load de ficheiros extensos. Desta forma, temos um script mais eficiente e cada vez que é preciso rodar a aplicação demora pouco tempo.

1. Inicialização e Leitura de Dados

```
clear all
clc

movies = readcell("movies.csv","Delimiter",',');

generos_totais = movies(1:end,3:end);

generos = unique(generos_totais(:,1));

titulos = movies(1:end,1);

n_titulos = length(titulos);

n_generos = length(generos_totais);

anos = movies(1:end,2);
```

- **clear all; clc;**

  Este comando remove todas as variáveis do Workspace e limpa a tela do Command Window. Achámos essencial para mantermos um ambiente limpo.

- **movies = readcell("movies.csv","Delimiter",',');**

  Aqui, tal como indicado no guião, executamos este comando para “ler” o arquivo CSV chamado "movies.csv". O arquivo é assumido como uma tabela com várias colunas, onde cada linha representa um filme e suas informações relacionadas.

- **Generos_totais = movies(1:end,3:end);**
- **generos = unique(generos(:,1));**
- **titulos = movies(1:end,1);**

- **anos = movies(1:end,2);**
  Generos_totais representa os géneros de todos os filmes; Generos representa um vetor com todos os géneros existentes, ou seja os conteúdos únicos a partir da 3ª coluna; Títulos contém os aproximados 58 mil títulos dos filmes (1ª coluna) e Anos contém o ano de cada filme existente.

2. Inicialização do Bloom Filter

- **Definição de Parâmetros e Inicialização**: O Bloom Filter é uma estrutura de dados eficiente para testar se um item está presente em um conjunto. Ele é usado aqui para armazenar géneros e combinações de géneros-ano. O parâmetro **k** define o número de funções hash utilizadas, o que afeta a precisão e a probabilidade de falsos positivos no Bloom Filter.

3. Preenchimento do Bloom Filter

- **Inserção de Dados**: O código insere dados nos Bloom Filters criados. Isso é feito para cada género de cada filme (BF_generos), e para cada combinação de género e ano (BF_generos_ano).

```
k = 4; % Número de funções de dispersão
BF_generos = bloomFilterInitialize(1000);
BF_generos_ano = bloomFilterInitialize(1000);
[n_linhas,n_colunas] = size(movies);

for i = 1:n_linhas
    for j = 3:n_colunas
        if ~ismissing(movies{i, j})
            BF_generos = bloomFilterInsert(BF_generos, movies{i, j}, k);
        end
    end
end

for i = 1:n_linhas
    ano = num2str(movies{i, 2});   % Segunda coluna para o ano
    for j = 3:n_colunas  % Gêneros começando na terceira coluna
        if ~ismissing(movies{i, j})
            genero = movies{i, j};
            chave = [genero ano]; % Criação da chave como 'Genero Ano'
            BF_generos_ano = bloomFilterInsert(BF_generos_ano, chave, k);
        end
    end
end
```

4. Processamento de Assinaturas para Títulos

Criação de Shingles e Assinaturas: Shingles são conjuntos de caracteres usados para representar partes de strings (neste caso, títulos de filmes). Isso é útil para comparar a similaridade entre strings. O script cria esses shingles e então calcula assinaturas, que são representações compactas desses shingles, usando uma função de hash (muxDJB31MA). Estas assinaturas são usadas para comparações eficientes entre títulos.

```
n_hash = 100;

set_titles = createShinglesTitle(titulos);

assinaturas_opcao4 = inf(n_titulos,n_hash);

for n = 1:n_titulos
    set_n = set_titles{n};
    for i = 1:length(set_n)
        key = num2str(set_n(i));
        h_out = muxDJB31MA(key, 127, n_hash);
        assinaturas_opcao4(n,:) = min(h_out, assinaturas_opcao4(n,:));
    end
end
```

5. Processamento de Assinaturas para Géneros

De forma similar ao passo anterior, criam-se shingles para os géneros de cada filme existente, e consequentemente a sua assinatura.

```
set_genres = createShinglesForGenres(generos_totais);

assinaturas_opcao5 = inf(n_generos,n_hash);

for n = 1:n_generos
    set_n = set_genres{n};
    for i = 1:length(set_n)
        key = num2str(set_n{i});
        h_out = muxDJB31MA(key, 127, n_hash);
        assinaturas_opcao5(n,:) = min(h_out, assinaturas_opcao5(n,:));
    end
end
```

6. Limpeza de Variáveis

Após o processamento, o script limpa variáveis não mais necessárias. Esta etapa é importante para economizar memória, evitar confusões com variáveis que não serão mais utilizadas e tempo de load dos dados guardados no próximo script.

7. Salvamento dos Dados

Por fim, o script guarda o estado atual das variáveis utilizadas no *workspace* do script1 em dados.mat, que é um formato de arquivo usado pelo MATLAB para armazenar variáveis. Desta forma, poderemos usar estes dados no próximo script.

```
clear n_colunas n_linhas ano chave ...
    genero k i j movies n_titulos shingle n_hash h_out key set_n n...
    set_genres set_titles n_generos

save dados.mat
```

8. Funções utilizadas no script 1

- **bloomFilterInitialize -** Inicialização de um filtro de Bloom de n tamanho.

```
function filtro = bloomFilterInitialize(n)
    filtro = zeros(n,1);
end
```

- **bloomFilterInsert –** A função realiza a inserção de um elemento em um filtro Bloom. Para isso, ela calcula k hashes do elemento (chave) e atualiza as posições correspondentes no filtro Bloom para indicar a presença da chave. Isso é feito através de um loop que itera k vezes, modificando ligeiramente a chave original, recalculando o hash e incrementando a posição do hash no filtro Bloom.

```
function bloom = bloomFilterInsert(bloom, key, k)
    m = length(bloom);
    aux = muxDJB31MA(key, 127, k);
    for i = 1:k
        key = [key num2str(i)];
        hash = mod(aux(i), m) + 1;
        bloom(hash) = bloom(hash) + 1;
    end
end
```

- **createShinglesTitle -** A função createShinglesTitle gera shingles para cada título de filmes. Shingles são combinações de caracteres usadas para representar dados de texto para comparações de similaridade. A função cria um conjunto de shingles de tamanho 2 para cada título e armazena-os em um *cell array.*

```
function [Set] = createShinglesTitle(titulos)

    n_titulos = length(titulos);
    Set = cell(n_titulos,1);

    for n = 1:n_titulos
        title = titulos{n};
        for i = 1:length(title)-1
            Set{n} = [Set{n} convertCharsToStrings(title(i:i+1))];
        end
    end
end
```

- **createShinglesForGenres -** A função createShinglesForGenres gera shingles para o(s) género(s) de cada filme.

```
function [Set] = createShinglesForGenres(generos_totais)
    Set = cell(size(generos_totais, 1), 1);

    for n = 1:size(generos_totais, 1)
        genreString = '';

        % Concatena todos os géneros válidos do filme n em uma única string
        for j = 1:size(generos_totais, 2)
            if isa(generos_totais{n,j}, 'char')
                genreString = [genreString, generos_totais{n,j}];
            end
        end

        % Cria shingles para todos os géneros concatenados
        shingles = {};
        for i = 1:length(genreString)-1
            shingle = genreString(i:i+1);
            shingles{end+1} = shingle;
        end

        Set{n} = shingles;
    end
end
```

- **muxDJB31MA -** A função muxDJB31MA calcula um conjunto de valores de hash para uma chave dada, utilizando o método DJB2 com um multiplicador de 31, uma técnica comum em funções de hash de strings.

```
function aux = muxDJB31MA(chave, seed, k)
    len = length(chave);
    chave = double(chave);
    h = seed;
    aux = zeros(1, k);
    for i=1:len
        h = mod(31 * h + chave(i), 2^32 -1) ;
    end
    for i = 1:k
        h = mod(31 * h + i, 2^32 -1) ;
        aux(i) = h;
    end
end
```

Script 2

O script inicia com um **clear all** e **clc**, com o intuito de, novamente, trabalharmos em um Workspace limpo e organizado A seguir é feito um load aos ficheiro onde estão armazenados os dados a utilizar neste script, ou seja, tudo o que se guardou no script anterior, irá ser usado neste script.

- **choice = 0;**: Inicializa a variável **choice** com o valor 0. Esta variável é utilizada para controlar o fluxo do menu interativo do script, permitindo ao usuário escolher entre várias opções.

1. Menu de Interação com o Usuário: O script inicia apresentando um menu interativo, com várias opções onde podem ser analisadas estatísticas sobre os filmes, incluindo visualização dos géneros existentes, contagem de filmes por género e ano, e procura de filmes similares a palavras introduzidas e/ou géneros.

```
clear all;
load dados.mat
clc
choice = 0;

while choice ~= 6
    fprintf('\n---------------------------------\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');
    fprintf('\n---------------------------------\n')
    choice = input('Select choice: ');
```

2. Escolha do Usuário e Processamento: Dependendo da escolha do usuário, o script executa diferentes funções. O loop **while** no script é usado para apresentar um menu interativo ao usuário até que a opção de sair (opção 6) seja escolhida. Dentro do loop, o script exibe várias opções, e o usuário seleciona uma delas digitando o número correspondente. O **switch** dentro do loop executa diferentes blocos de código baseados na escolha do usuário.

3. Análise de cada opção do menu:

1. **Display Available Genres**: Lista todos os géneros de filmes disponíveis no banco de dados.

```
case 1
    fprintf("\n-- List of available genres --\n")
    for i = 1:length(generos)
        fprintf('%d. %s\n', i, generos{i});
    end
```

É feita a listagem dos géneros existentes. A variável **géneros** tinha sido criada no script anterior. A lógica desta variável foi explicada anteriormente na parte do relatório relativa ao script1.

2. **Number of Movies of a Genre**: Permite ao usuário selecionar um género e exibe a quantidade de filmes desse género.

```
case 2
    genre = input('Select a genre: ', 's');
    if verificaGeneroNaLista(genre, generos) == false
        fprintf("\nGénero inválido!")
    else
        n_filmes_genero = bloomFilterCheck(BF_generos,genre,4);
        fprintf('\nNumber of movies in the genre %s: %d\n', genre, n_filmes_genero);
    end
```

Na "Opção 2" do menu, o código solicita ao usuário que insira um género e armazena a entrada na variável **genre**. Chama a função **verificaGeneroNaLista** para verificar se o género inserido está presente na lista de géneros disponíveis.

Esta função funciona da seguinte forma: A função **strcmp** compara o género inserido com cada género na lista de géneros existentes. A função **any** é usada para determinar se alguma das comparações retornou verdadeira, indicando que o género está presente na lista. Se encontrado, **genero_valido** será **true**; caso contrário, será **false**.

```
function genero_valido = verificaGeneroNaLista(genero, lista_generos)
    genero_valido = any(strcmp(genero, lista_generos));
end
```

Se o género não estiver na lista, imprime uma mensagem de erro e o menu volta a ser exibido. Caso contrário, chama a função **bloomFilterCheck** para verificar o número de filmes desse género usando um filtro Bloom, que depois imprime esse valor (count).

```
function count = bloomFilterCheck(bloom, key, k)
    m = length(bloom);
    key = convertStringsToChars(key);
    aux = muxDJB31MA(key, 127, k);
    count = [];
    for i = 1:k
        key = [key num2str(i)];
        hash = mod(aux(i), m) + 1;
        if bloom(hash) > 0
            count = [count bloom(hash)];
        end
    end
    count = min(count); %slides
end
```

3. **Number of Movies of a Genre in a Given Year**: O usuário insere um género juntamente com um ano, e o script mostra quantos filmes desse género foram lançados naquele ano.

```
case 3
    genero_e_ano = input("Select a genre and a year (separated by ','): ","s");
    genero_e_ano = strsplit(genero_e_ano,',');
    if length(genero_e_ano) ~= 2
       fprintf("\nInput inválido!\n")

    else
        genre = string(genero_e_ano{1});

        year = genero_e_ano{2};


        if verificaGeneroNaLista(genre,generos)==false || str2double(year) < 0 || str2double(year)  > 2023
           fprintf("\nInput inválido!")

        else
           key = strcat(genre,',',year);
           n_filmes_genero_ano = bloomFilterCheck(BF_generos_ano,key,4);
           fprintf('\nNumber of movies in the genre %s and from year %d: %d\n', genre, str2double(year),n_filmes_genero_ano);

        end
    end
```

Recolha e Validação de Entrada:

O utilizador é solicitado a inserir um género e um ano, separados por vírgula. A string inserida é dividida em duas partes usando a função **strsplit**. Se a entrada não contiver exatamente duas partes após a divisão, uma mensagem de "Input inválido!" é impressa.

Processamento e Criação de Chave:

Se a entrada for válida, ela será processada para extrair o género e o ano.

É realizada uma verificação para confirmar se o género existe na lista **generos** e se o ano é um número válido dentro do intervalo.

Verificação no Filtro de Bloom:

É criada uma chave concatenando o género e o ano, separados por uma vírgula. Essa chave é usada para verificar no filtro de Bloom (**bloomFilterCheck**) se há filmes que correspondem ao género e ano especificados. A função **bloomFilterCheck** retorna o número de filmes que correspondem a essa chave.

Impressão do Resultado:

Por fim, imprime o número de filmes que correspondem ao género e ano fornecidos pelo utilizador, utilizando o valor retornado pelo **bloomFilterCheck**.

4. **<u>Search Movie Titles</u>**: O usuário digita uma string e o script busca por títulos de filmes similares, usando MinHash para comparar a similaridade.

```
case 4
    user_string = input("Insert a string: ","s");

    n_hash = 100;
    Set = cell(1,1);
    user_minhash_signature = inf(1,n_hash);

    for i = 1:length(user_string)-1
        Set{1} = [Set{1} convertCharsToStrings(user_string(i:i+1))];
    end


    for i = 1:length(Set{1})
        key = Set{1}{i};
        h_out = muxDJB31MA(key, 127, n_hash);
        user_minhash_signature = min(h_out, user_minhash_signature);
    end

    similarities = zeros(size(assinaturas_opcao4, 1), 1);

    for i = 1:size(assinaturas_opcao4, 1)
        similarities(i) = sum(assinaturas_opcao4(i, :) == user_minhash_signature) / n_hash;
    end
```

Preparação para MinHash:

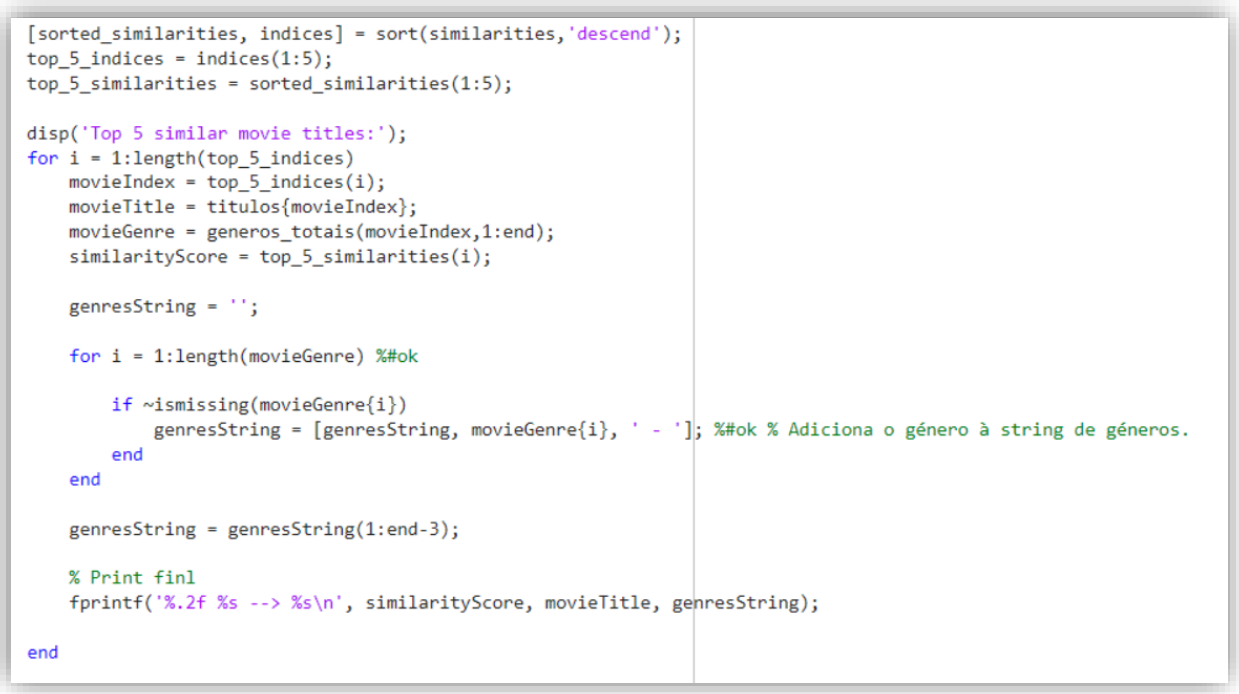
É definido um número de funções hash (n_hash = 100). É inicializado um conjunto (Set) para armazenar substrings de dois caracteres da string inserida. A assinatura MinHash do usuário é inicializada com infinito (inf), indicando que ainda não foram encontrados valores mínimos.

Criação de Shingles e Cálculo da Assinatura MinHash:

A string inserida pelo usuário é dividida em substrings de dois caracteres (shingles) e armazenada em Set. Para cada shingle, é calculada uma assinatura MinHash usando a função muxDJB31MA. A assinatura MinHash do usuário é atualizada com o menor valor entre o hash atual e o valor existente na assinatura.

Cálculo de Similaridade:

As similaridades entre a assinatura MinHash do usuário e as assinaturas preexistentes (assinaturas_opcao4) são calculadas.



```
[sorted_similarities, indices] = sort(similarities,'descend');
top_5_indices = indices(1:5);
top_5_similarities = sorted_similarities(1:5);

disp('Top 5 similar movie titles:');
for i = 1:length(top_5_indices)
    movieIndex = top_5_indices(i);
    movieTitle = titulos{movieIndex};
    movieGenre = generos_totais(movieIndex,1:end);
    similarityScore = top_5_similarities(i);

    genresString = '';

    for i = 1:length(movieGenre) %#ok

        if ~ismissing(movieGenre{i})
            genresString = [genresString, movieGenre{i}, ' - ']; %#ok % Adiciona o género à string de géneros.
        end
    end

    genresString = genresString(1:end-3);

    % Print finl
    fprintf('%.2f %s --> %s\n', similarityScore, movieTitle, genresString);

end
```

Ordenação e Seleção dos Top 5:

As similaridades são ordenadas em ordem decrescente. Os índices dos 5 filmes mais similares são armazenados em top_5_indices. As 5 maiores similaridades são armazenadas em top_5_similarities.

Exibição dos Resultados:

Os 5 títulos de filmes mais similares são exibidos. Para cada filme, os géneros associados são concatenados em uma string, separados por um traço (-). De seguida, o último traço e espaço são removidos. A similaridade, o título do filme e a string de géneros são impressos.

5. **<u>Search Movies Based on Genres</u>**: Permite ao usuário inserir um ou mais géneros e o script lista filmes correspondentes a esses géneros.

```
case 5
    continuar = false;
    while ~continuar
        palavras_introduzidas = input("\nSelect one or more genres (separated by ','): ", "s");
        palavras_introduzidas = strsplit(palavras_introduzidas, ',');
        for i = 1:length(palavras_introduzidas)
            if verificaGeneroNaLista(string(strtrim(palavras_introduzidas{i})), generos) == false
                fprintf('\nOne or more entered genres are not valid. Please try again.\n');
                continuar = false;
                break;
            else
                continuar = true;
            end
        end
    end

    string_palavras_introduzidas = "";
    for i = 1:length(palavras_introduzidas)
        string_palavras_introduzidas = strcat(string_palavras_introduzidas, palavras_introduzidas{i});
    end
    string_palavras_introduzidas = convertStringsToChars(string_palavras_introduzidas);

    n_hash = 100;
    Set = cell(1,1);
    user_minhash_signature = inf(1,n_hash);

    for i = 1:length(string_palavras_introduzidas)-1
        Set{1} = [Set{1} convertCharsToStrings(string_palavras_introduzidas(i:i+1))];
    end

    for i = 1:length(Set{1})
        key = Set{1}{i};
        h_out = muxDJB31MA(key, 127, n_hash);
        user_minhash_signature = min(h_out, user_minhash_signature);
    end

    similarities = zeros(size(assinaturas_opcao5, 1), 1);
    for i = 1:size(assinaturas_opcao5, 1)
        similarities(i) = sum(assinaturas_opcao5(i, :) == user_minhash_signature) / n_hash;
    end
```

Validação de Géneros:

O usuário insere um ou mais géneros separados por vírgula. Os géneros são divididos e verificados um a um para garantir que estão na lista de géneros permitidos (generos). Se algum género não for válido, o usuário é solicitado a inserir novamente os géneros.

Criação da Assinatura MinHash:

Os géneros válidos são concatenados em uma única string (string_palavras_introduzidas). Cada par de caracteres consecutivos na string concatenada é adicionado a um conjunto (Set). Para cada elemento de Set, é calculada uma assinatura MinHash usando muxDJB31MA. A

assinatura MinHash do usuário é determinada pelo menor valor hash encontrado para cada função hash.

Cálculo de Similaridades:

As similaridades entre a assinatura MinHash do usuário e as assinaturas de assinaturas_opcao5 são calculadas. As similaridades são armazenadas em um vetor e ordenadas em ordem decrescente.

```
[sorted_similarities, indices] = sort(similarities,'descend');
top_5_indices = indices(1:5);
top_5_similarities = sorted_similarities(1:5);

ordenados = cell(5,3);

for i = 1:length(top_5_indices)
    movieIndex = top_5_indices(i);

    ordenados{i, 1} = titulos{movieIndex}; %Título do filme

    ordenados{i, 2} = top_5_similarities(i); %Similaridade

    ordenados{i, 3} = anos{movieIndex}; %Ano do filme
end

ordenados=sortrows(ordenados, [2,3], 'descend');

for i = 1:5
    fprintf('%.2f %s --> %d\n', ordenados{i,2}, ordenados{i,1}, ordenados{i,3})
end
```

Seleção e Ordenação dos Top 5:

Os índices dos 5 filmes mais similares são armazenados em top_5_indices.

As 5 maiores similaridades são armazenadas em top_5_similarities.

As informações são organizadas no cell array ordenados, que inclui o título do filme, a similaridade e o ano. Este array é então ordenado pela similaridade e pelo ano no caso de a similaridade ser a mesma, ambos em ordem decrescente.

Exibição dos Resultados:

Os 5 filmes mais similares são impressos, exibindo a similaridade, o título e o ano do filme associado.

6. **Exit**: Encerra o programa.

Conclusão

Com a realização deste guião conseguimos colocar em prática tudo que nos foi ensinado nas aulas práticas, aprofundamos conhecimentos sobre a matéria e ficamos mais proficientes com os algoritmos em causa.

Para finalizar, podemos dizer que conseguimos alcançar grande parte dos objetivos propostos e consideramos ter tido um resultado satisfatório visto o tempo dado para a execução do mesmo.