deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

Relatório Orientado

Guião PL04

Métodos Probabilísticos para Engenharia Informática

Departamento de Eletrónica, Telecomunicações e Informática

Prof. Amaro Sousa

Ano letivo 2020/2021

Turma P3

André Pragosa Clérigo, 98485

Tiago Afonso Marques, 98459

23/01/2021

Índice

Introdução

Começamos por fazer 2 scripts para o funcionamento desta aplicação, a um deles chamámos de “database.m” onde temos código que apenas é preciso ser executado periodicamente caso haja uma mudança nos filmes disponíveis no vídeo clube ou caso um utilizador já tenha visto mais filmes. Este código armazena o nome dos filmes e as suas categorias, armazena os filmes vistos por cada utilizador, implementa a função MinHash para cada título e para cada shingle do mesmo. Ao outro script chamámos de “app.m” que vai correr a app em si e dá *load* às variáveis que foram criadas na “database.m”

Considere que:

- O ficheiro “u.data” e “u_item.txt” estão no mesmo diretório dos scripts para o guião;
- Algum código é completamente reutilizado, logo código usado mais do que uma vez não irá ser comentado novamente;

Database

```
dic = readcell('u_item.txt');

udata = load('u.data');
u = udata(1:end,1:2);
clear udata;

users = unique(u(:,1)); % Extrai os IDs dos utilizadores
Nu = length(users); % Numero de utilizadores

Set = cell(Nu,1); % Usa celulas
for n = 1:Nu % Para cada utilizador
    % Obtem os filmes de cada um
    ind = find(u(:,1) == users(n));
    % E guarda num array. Usa celulas porque utilizador tem um numero
    % diferente de filmes. Se fossem iguais podia ser um array
    Set{n} = [Set{n} u(ind,2)];
end

K = 100;  % Número de funções de dispersão
MinHashValue = inf(Nu,K);
for i = 1:Nu
    conjunto = Set{i};
    for j = 1:length(conjunto)
        chave = char(conjunto(j));
        hash = zeros(1,K);
        for kk = 1:K
            chave = [chave num2str(kk)];
            hash(kk) = DJB31MA(chave,127);
        end
        MinHashValue(i,:) = min([MinHashValue(i,:); hash]);  % Valor
minimo da hash para este título
    end
end

shingle_size=3;
K = 150;  % Número de funções de dispersão
MinHashSig = inf(length(dic),K);
for i = 1:length(dic)
    conjunto = lower(dic{i,1});
    shingles = {};
    for j= 1 : length(conjunto) - shingle_size+1  % Criacao dos
shingles para cada filme
        shingle = conjunto(j:j+shingle_size-1);
        shingles{j} = shingle;
    end

    for j = 1:length(shingles)
        chave = char(shingles(j));
        hash = zeros(1,K);
        for kk = 1:K
            chave = [chave num2str(kk)];
            hash(kk) = DJB31MA(chave,127);
        end
        MinHashSig(i,:) = min([MinHashSig(i,:);hash]);  % Valor minimo
da hash para este shingle
    end
end
save 'database' 'dic' 'Nu' 'users' 'Set' 'MinHashValue' 'MinHashSig'
```

Para decidirmos o valor de K para a MinHash dos nomes dos filmes fomos rever o exercício 4 da secção 4.3 do último guião prático, reparamos que para K = 50, 100 e 200 os tempos de calculo de MinHash são 71, 167 e 408 segundos respetivamente, vermos também que o cálculo das distâncias de Jaccard para todos os pares possíveis de utilizadores é demoram 9, 2 e 3 segundos respetivamente e com tempos de cálculo do par mais similar bastante semelhantes entre todos. Com estes resultados obtidos e ao ver qual o verdadeiro valor da distância de Jaccard entre 2 pares de utilizadores, achamos por bem utilizar o valor de K = 100, vistos que este script apenas irá ser executado periodicamente. Ficamos assim com valores de similaridade mais próximos do valor real e fazendo com que o cálculo das distâncias no script "app.m" seja mais rápido.

Ao variar o tamanho do shingle entre 2 e 5 vemos que o cálculo de todos os shingles possíveis para todos os filmes varia entre os 0,7 segundos para um size de 2 até 0,3 segundos para um size de 5, chegando à conclusão de que pelo menos para esta fase na criação dos shingles o tamanho não afeta a nossa aplicação.

Para decidir o valor K para a MinHash dos shingles voltamos a correr valores de K para 50, 100, 150 e 200 para ver quanto tempo o código demoraria obtendo 25, 53, 90 e 134 segundos respetivamente. Tal como na função MinHash anterior o balanço é muito simples, um valor de K maior vai consequentemente dar um valor mais exato para o cálculo das distâncias de Jaccard mas demorando mais tempo a fazer as suas MinHash, decidimos então ficar pelo valor K = 150, tendo em conta que este valor irá ser usado para fazer a MinHash da *string* introduzida pelo utilizador e vistos que esta não tem um tamanho máximo, valores elevados de K podem fazer com que este processo seja mais demorado do que o desejado. Não obstante, consideramos que para um K = 150 as MinHash produzidas são precisas o suficiente para obter resultados satisfatórios para a função **searchTitle**.

App

```
clc;
clear all;

load database; %'dic' 'Nu' 'users' 'Set' 'MinHashValue' 'MinHashSig'

user = 0;
option = 0;

menu(user, option, Nu, Set, dic, MinHashValue);
```

Apenas damos *load* aos valores calculados na "database.m" e chamamos a função **Menu**

Menu

```
function menu(user, option, Nu, Set, dic, MinHashValue, MinHashSig)
    while(isempty(option) | option < 4 | user == 0)
        clc
        if (user == 0)
            user = str2num(input(['Insert User ID (1 to ' num2str(Nu)
'): '], 's'));
        elseif (isempty(user) || user< 1 || user>Nu)
            fprintf(2, 'User ID not valid.');
            fprintf(2, '\nPress any key to continue. ');
            pause; clc;  % Manter a informação disponível até ao
utilizador pressionar em qualquer tecla
            user = 0;
        else
            fprintf('\nUser ID: %d\nMenu', user);
            fprintf('\n1 - Your Movies\n2 - Get Suggestions\n3 -
Search Title\n4 - Exit\nSelect choice: ');
            option = str2num(input('', 's'));
            if isempty(option)
                continue;
            end

            switch option
                case 1
                    yourMovies(user, Set, dic);
                case 2
                    getSuggestions(Nu, MinHashValue, user, Set, dic);
                case 3
                    searchTitle(dic, MinHashSig)
                case 4
                    clc
                    break;
                otherwise
                    option = 0;
                    clc
            end
        end
    end
end
```

A função **menu** é a "porta de entrada" da nossa app, onde o utilizador terá de escolher um user e consequentemente a operação/operações que este deseja realizar. Aceita como parâmetros o ID do user escolhido (user) e a option que corresponde ao número da operação a realizar, ambos estes valores inicialmente são 0. O Set que contem todos os ids dos filmes vistos por todos os utilizadores, o dic que contem todos os filmes disponíveis e as suas categorias, o número de users disponíveis (Nu) e a MinHashValue que contém todos os valores de dispersão calculados.

Fizemos os testes dentro do possível de modo que quando é pedido ao utilizador um ID ou uma opção, este é capaz de colocar qualquer tipo de input de modo a não dar *crash* à aplicação e apenas aceitar os valores válidos. Com o valor inserido não é válido o ecrã é limpo e o menu anterior volta a ser impresso.

Função yourMovies

```
function yourMovies(user, Set, dic)
    fprintf('\nYour movies watched:\n');
    for i = 1:length(Set{user})
        fprintf('%d - %s\n', i, dic{Set{user}(i)});
    end
    fprintf(2, 'Press key to continue. ');
    pause; clc;  % Manter a informação disponível até ao utilizador
pressionar em qualquer tecla
end
```

A função **yourMovies** tem como objetivo imprimir uma lista dos filmes que o utilizador escolhido previamente já viu. Aceita como parâmetros o ID do user escolhido (user), o Set que contem todos os ids dos filmes vistos por todos os utilizadores e o dic que contem todos os filmes disponíveis e as suas categorias.

Função getSuggestions

```
function getSuggestions(Nu, MinHashValue, user, Set, dic)
    while true
        fprintf('\nSelect the type of Movie\n1- Action, 2- Adventure,
3- Animation, 4- Children's, 5- Comedy, 6- Crime, 7- Documentary, 8-
Drama, 9- Fantasy\n10- Film-Noir, 11- Horror, 12- Musical, 13-
Mystery, 14- Romance, 15- Sci-Fi, 16- Thriller, 17- War, 18-
Western\nSelect choice: ');
        type = str2num(input('', 's'));
        if(type >= 1 && type <= 18)
            type = type + 2;
            break;
        end
        fprintf(2, 'Invalid number! ');
        pause; clc; % Manter a informação disponível até ao utilizador
pressionar em qualquer tecla
    end

    K = 100;  % Usamos o mesmo número de funcoes de dispersão usados
para a MinHash na database
    J = ones(1, Nu); % array para guardar distâncias
    h = waitbar(0, 'Calculating');
    for n = 1:Nu
        waitbar(n/Nu, h);
        if n ~= user
            J(n) = sum(MinHashValue(n,:) ~= MinHashValue(user,:))/K;
% Calculamos a distancia de Jaccard para todos os pares possiveis
desse user
        end
    end
    delete(h);
    [val, SimilarUserId] = min(J);

    suggestions = [];
    for n = 1:length(Set{SimilarUserId})
        % Se o similarUser tiver algum filme vistos que o user nao
viu, e
        % for da categoria escolhida, esse é um filme que vai ser
sugerido
        if (~ismember(Set{SimilarUserId}(n), Set{user}) &&
dic{Set{SimilarUserId}(n), type} == 1)
            suggestions = [suggestions
string(dic{Set{SimilarUserId}(n), 1})];
        end
    end

    if isempty(suggestions)  % Se nao houver sugestoes
        fprintf('\nThere is no film suggestions.\n');
    else
       fprintf('\nFilm Suggestions:\n');
       for i = 1:length(suggestions)  % Display dos filmes sugeridos
           fprintf(suggestions(i) + '\n');
       end
    end
    fprintf(2, 'Press any key to continue. ');
    pause;clc;  % Manter a informação disponível até ao utilizador
pressionar em qualquer tecla
end
```

A função **getSuggestions** tem como objetivo sugerir ao *user* escolhido inicialmente uma seleção de filmes da categoria escolhida por ele (ação, comedia, etc). Aceita como parâmetros o ID do user escolhido (*user*), o Set que contém todos os ids dos filmes vistos por todos os utilizadores, o dic que contém todos os filmes disponíveis e as suas categorias, o número de *users* disponíveis (Nu) e a MinHashValue que contém todos os valores de dispersão calculados.

Com o valor das MinHash fazemos as distâncias de Jaccard usando o mesmo K (100) que foi usado na nossa "database.m", de notar que iniciamos o vetor com o valor 1 para quando a distância de Jaccard for calculada entre o utilizador e ele mesmo o valor da distância ser 1 e assim nunca conseguirá ser o valor mínimo desse vetor. Obtendo o id do utilizador mais similar usamos a função *ismember* para verificar se há algum filme que o utilizador similar viu que o nosso *user* não tenha. Caso esse filme seja do tipo selecionado, esse filme vai ser sugerido.

Função searchTitle

```
function searchTitle(dic, MinHashSig)
    str = lower(input('\nWrite a String: ', 's'));
    shingle_size = 3;  % Utilizar o mesmo numero de shingles que na
database
    K = size(MinHashSig, 2);  % Usar o K igual ao K utilizado na base
de dados para os shingles dos titulos
    threshold = 0.99;  % Definir um threshold que nos é dado

    % Cell array com os shingles da string introduzida
    shinglesAns = {};
    for i = 1:length(str) - shingle_size+1
        shingle = str(i:i+shingle_size-1);
        shinglesAns{i} = shingle;
    end

    % Fazer a MinHash dos shingles da string introduzida
    MinHashString = inf(1,K);
    for j = 1:length(shinglesAns)
        chave = char(shinglesAns{j});
        hash = zeros(1,K);
        for kk = 1:K
            chave = [chave num2str(kk)];
            hash(kk) = DJB31MA(chave, 127);
        end
        MinHashString(1,:) = min([MinHashString(1,:); hash]);
    end

    % Distancia de Jaccard entre a string e cada filme
    distJ = ones(1, size(dic,1));  % array para guardar distancias
    h = waitbar(0,'Calculating');
    for i=1:size(dic, 1)  % cada hashcode da string
        waitbar(i/K, h);
        distJ(i) = sum(MinHashSig(i,:) ~= MinHashString)/K;
    end
    delete(h);

    flag = false;  % Fazemos uma flag para saber se houve algum filme
encontrado com uma distancia menor ou igual a threshold
    for i = 1:5
        [val, pos] = min(distJ);  % Calcular o valor minimo (mais
similaridade)
        if (val <= threshold)  % Se o valor minimo já nao pretencer ao
threshold não dá print
            flag = true;
            fprintf('%s   (%f)\n', dic{pos, 1}, val);
        end
        distJ(pos) = 1;  % Retirar esse filme dando uma distancia
igual a 1
    end

    if (~flag)
        fprintf('No movies found.\n');
    end
    fprintf(2, 'Press any key to continue. ');
    pause;clc;  % Manter a informação disponível até ao utilizador
pressionar em qualquer tecla
end
```

Na função **searchTitle** tivemos uma aproximação simplista, em que dado a *string* que o utilizador introduz apenas criamos os seus shingles e fazemos a respetiva MinHash para cada shingle, de seguida usamos o método que foi usado na função **getSuggestions** para calcular as distâncias de Jaccard entre os shingles da *string* introduzida e os shingles dos filmes. Depois disso vemos qual o valor mínimo das distâncias de Jaccard e enquanto esse valor mínimo for menor ou igual que o *threshold* definido, fazemos o display do nome desse filme. Caso nenhuma distância seja menor do que o *threshold* definido dizemos que não foram encontrados filmes. Para manter a ordem do filme mais similar, damos o valor de 1 à distância de Jaccard do filme que foi exibido, evitando alterações na matriz e fazendo com que esse filme nunca mais seja selecionado como o mínimo.

A nossa função aceita qualquer tipo de *string* com qualquer tamanho.

Nota Final

Em todo este programa usamos a função dada “DJB31MA.m”, função que nos foi dada no início deste guião e que a sua eficácia foi comparada com outras várias funções de *hash*. Com a realização do guião PL04 decidimos que esta seria a função mais indicada para realizar as nossas funções MinHash. De notar também que não usamos as funções de dispersão dentro da função DJB31MA, ou seja, trocamos a maior eficácia de código (fazer a adição do número K no final da *string* a ser hashada), pela conveniência de podermos mudar o valor de K utilizado na nossa “app.m” e “database.m” de modo a que obtermos resultados mais rápidos e menos precisos, ou resultados mais precisos mas que demorem mais tempo a correr, dependendo das necessidades do utilizador do nosso programa.

Para melhorarmos a nossa função **searchTitle** alterámos os shingles dos filmes que foram feitos na “database.m” e os shingles da *string* para ficarem com o seu conteúdo em minúsculas.