deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

Relatório

Guião PL04

Métodos Probabilísticos para Engenharia Informática

Departamento de Eletrónica, Telecomunicações e Informática

Ano letivo 2023/2024

Turma P8

Guilherme Santos, 107961

João Gaspar, 107708

21/12/2023

Índice

Introdução

Começamos por fazer 2 scripts para o funcionamento desta aplicação, a um deles chamámos de *data.m* onde temos código que apenas é preciso ser executado periodicamente caso haja uma mudança no ficheiro *restaurantes.txt* ou/e *utilizadores.data*. Ao outro script chamámos de *main.m* que vai correr a app em si.

Considere que:

- O ficheiro *restaurantes.txt* e *utilizadores.data* estão no mesmo diretório dos scripts para o guião;
- Foram criados outros ficheiros de funções individuais que são necessários para a execução do código;
- Algum código é completamente reutilizado, logo, código usado mais do que uma vez, não irá ser comentado novamente.

data.m

```
file = load("utilizadores.data");
rest = readcell('restaurantes.txt', 'Delimiter','\t');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% OPCAO 2
userIDs_uni = unique(file(:,1))';
Nu = length(userIDs_uni);
Set = cell(Nu,1);
userIDs = (file(:,1))';

k2 = 100;

h = waitbar(0, "Creating Set");
for n = 1:Nu
    waitbar(n/Nu,h);
    for i = 1:length(userIDs)
        if userIDs_uni(n) == userIDs(i)
            Set{n} = [Set{n} file(i,2)];
        end
    end
end
delete(h);

MA_2 = inf(Nu,k2);

h = waitbar(0, "Creating MA_2");
for i = 1:Nu
    waitbar(i/Nu,h);
    conjunto = Set{i};
```

```
    for j = 1:length(conjunto)
        elemento = char(conjunto(j));
        hash = zeros(1,k2);

        for n = 1:k2
            elemento = [elemento num2str(n)];
            hash(n) = DJB31MA(elemento,127);
        end

        MA_2(i,:)=min([MA_2(i,:);hash]);
    end
end
delete(h);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% OPCAO 3
shingle_size=3;
K = 100;
MinHashSig = inf(length(rest),K);
for i = 1:length(rest)
    conjunto = lower(rest{i,6});
    shingles = cell(1, length(conjunto) - shingle_size + 1);
    for j= 1 : length(conjunto) - shingle_size+1
        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]);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% OPCAO 4
restauranteIDs = rest(:,1);
numRest = length(restauranteIDs);
restAval = zeros(numRest,2);

userIDs = (file(:,1))';

for i = 1:numRest
    restAval(i,1) = restauranteIDs{i};
    avalT = 0;
    c = 0;
    for j = 1:length(userIDs)
        if restauranteIDs{i} == file(j,2)
            c = c + 1;
            avalT = avalT + file(j,4);
        end
    end
    if c == 0
```

```
        restAval(i,2) = 0;
    else
        restAval(i,2) = avalT / c;
    end
end

Nr = length(restauranteIDs);
Conjuntos = cell(Nr, 1);

restNM = rest;
for r = 1:Nr
    for i = 1:7
        if ismissing(rest{r,i})
            restNM{r,i} = "";
        end
    end
end

for n = 1:Nr
    Conjuntos{n} = [restNM(n, 3) restNM(n, 4) restNM(n, 5)  restNM(n, 6)
restNM(n, 7)];
end

k4 = 150;
nc = length(Conjuntos);
M = zeros(k4, nc);

for nu = 1:nc
    C = Conjuntos{nu};
    temp = inf(k4, 1);
    for j = 1:length(C)
        chave = char(C{j});
        hash = zeros(k4, 1);

        for kk = 1:k4
            chave_temp = [chave num2str(kk)];
            hash(kk) = DJB31MA(chave_temp, 127);
        end
        temp = min(temp, hash);
    end
    M(:, nu) = temp;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% OPCAO 5
userIDs = (file(:,1))';
n = 10000;
B = initFBC(n);

k = 3;
for i = 1:length(userIDs)
    B = addElemFBC(B,userIDs(i),k);
end

save("dados.mat", "B", "k", "MA_2", "k2", "MinHashSig", "M", "k4",
"restAval")
```

Para decidirmos os valores de k para a MinHash, fomos rever o exercício 4 da seção 4.3 do último guião prático e reparámos que para k = 50, 100 e 200, os tempos de cálculo de MinHash são 71, 167 e 408 segundos respetivamente, vimos também que o cálculo das distâncias de Jaccard para todos os pares possíveis 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, achámos que seria melhor utilizar o valor de 100 para k visto que este script será executado periodicamente. Ficamos assim com valores de similaridade mais próximos do valor real e fazendo com que o cálculo das distâncias no script *main.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 pratos 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 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.

Para a opção 5, nós decidimos inicializar o Filtro de Bloom de contagem com um tamanho de 10000, permitindo através deste valor uma redução de falsos positivos. Através da função *addElemFBC* adicionamos a quantidade de vezes que um utilizador fez uma avaliação. Decidimos utilizar um k, número de Hash Functions, com valor de 3, pois sendo m, o número de utilizadores distintos, e sendo n o tamanho do Filtro de Bloom de contagem, um k ótimo será: 0.693 * n / m = 0.693 * 10000 / 2426 = 2.86.

main.m

```
clear;
clc;

rest = readcell('restaurantes.txt', 'Delimiter', '\t');
file = load("utilizadores.data");
userIDs = (file(:,1));
```

```
data = load("dados.mat");
id = 0;
```

Inicializamos variáveis com os dados presentes nos ficheiros fornecidos para o guião e damos *load* aos valores que estão no ficheiro *dados.mat*.

```
while(1)
    clc;
    if (id == 0)
        tmp = input('Insert User ID (1 to ??): ');
        if ~ismember(tmp, userIDs)
            fprintf('Invalid User ID.');
            pause(2);
        else
            id = tmp;
        end
    else
      …
```

É iniciado um loop infinito que pede ao utilizador que insira um ID de usuário; o loop continuará até que o ID seja válido.

```
      x = input(['1 - Restaurants evaluated by you' ...
                        '\n2 - Set of restaurants evaluated by the most
      similar user' ...
                        '\n3 - Search special dish' ...
                        '\n4 - Find most similar restaurants' ...
                        '\n5 - Estimate the number of evaluations by each
      tourist' ...
                        '\n6 - Exit' ...
                        '\nSelect choice: ']);
        switch x
            case 1
                restaurantYouEvaluated(rest, file, userIDs, id);
                fprintf('\nPress any key to go back.');
                pause;
            case 2
                similarRestaurant(file, rest, data.MA_2, data.k2, id,
userIDs);
                fprintf('\nPress any key to go back.');
                pause;
            case 3
                searchPlate(rest, data.MinHashSig);
                fprintf('\nPress any key to go back.');
                pause;
            case 4
                selectRestaurant(rest, file, userIDs, id, data.M, data.k4,
data.restAval);
                fprintf('\nPress any key to go back.');
                pause;
            case 5
                tmp = input('Insert User ID: ');
                if ~ismember(tmp, userIDs)
```

```
                    fprintf('Invalid User ID.');
                    pause(2);
                else
                    id = tmp;
                    number = countFBC(data.B,id,data.k);
                    fprintf('Number of evaluations made by User ID %d: %d
\n', id, number);
                    fprintf('\nPress any key to go back.');
                    pause;
                end
            case 6
                return
            otherwise
                disp('Invalid option.\n')
        end
    end
end
```

Aqui o utilizador terá de escolher qual operação/operações que deseja realizar.

restaurantYouEvaluated

```
function restaurantYouEvaluated(rest, file, userIDs, id)
    restaurentes = zeros(1, length(userIDs));
    rlen = 1;
    for i = 1:length(userIDs)
        if userIDs(i) == id
            restaurentes(rlen) = file(i,2);
            rlen = 1 + rlen;
        end
    end

    restIDs = rest(:,1)';
    restName = rest(:,2)';
    restLocal = rest(:,3)';
    for i = 1:length(restIDs)
        if ismember(restIDs{i},restaurentes)
            fprintf("ID: %d; Nome: %s; Localidade: %s.\n", restIDs{i},
restName{i}, restLocal{i})
        end
    end
end
```

Esta função tem como objetivo imprimir uma lista dos ID, nomes e localidades de restaurantes que o utilizador já avaliou. Aceita como parâmetros o *rest* que contém todos os detalhes dos restaurantes, o *file* que contém os utilizadores, os *userIDs* e o *id* do utilizador que iniciou o programa e o determinou.

similarRestaurant

```
function similarRestaurant(file, rest, MA_2, k2, id, userIDs)
```

```
    userIDs_uni = unique(file(:,1))';
    Nu = length(userIDs_uni);

    J=zeros(Nu,Nu);

    for n1= 1:Nu
        for n2= n1+1:Nu
            J(n1,n2) = sum(MA_2(n1,:)==MA_2(n2,:))/k2;
            J(n2,n1) = sum(MA_2(n1,:)==MA_2(n2,:))/k2;
        end
    end

    SimilarUsers= zeros(1,3);

    for n1= 1:Nu
        if userIDs_uni(n1) == id
            for n2= 1:Nu
                if J(n1,n2) > SimilarUsers(1,3)
                    SimilarUsers(1,:)= [userIDs_uni(n1) userIDs_uni(n2)
J(n1,n2)];
                end
            end
        end
    end
    id2 = SimilarUsers(1,2);

    restaurentes = [];
    rlen = 1;
    for i = 1:length(userIDs)
        if userIDs(i) == id2
            restaurentes(rlen) = file(i,2);
            rlen = 1 + rlen;
        end
    end

    restIDs = rest(:,1)';
    restName = rest(:,2)';
    restLocal = rest(:,3)';
    fprintf("User ID: %d\n",id2)
    for i = 1:length(restIDs)
        if ismember(restIDs{i},restaurentes)
            fprintf("ID: %d; Nome: %s; Localidade: %s.\n", restIDs{i},
restName{i}, restLocal{i})
        end
    end
end
```

Esta função tem como objetivo mostrar ao utilizador os restaurantes avaliados pelo utilizador mais parecido a ele mesmo. Aceita como parâmetros o *file* que contém os utilizadores, o *rest* que contém todos os detalhes dos restaurantes, o *MA_2* que provém do *data.m* e que representa uma matriz de valores hash mínimos para cada utilizador, o *k2* que também provém do *data.m* e representa o número de funções de dispersão, o *id* e o *userIDs*.

Com o valor das MinHash fazemos as distâncias de Jaccard usando o mesmo *k2* (100) que foi usado na *data.m*. Obtendo o *id* do utilizador mais similar usamos a função *ismember* para verificar se há algum restaurante que o utilizador mais similar avaliou e assim, imprimi-lo.

searchPlate

```
function searchPlate(rest, MinHashSig)
    str = lower(input('Write a String: ', 's'));
    shingle_size = 3;
    K = size(MinHashSig, 2);
    threshold = 0.99;

    shinglesAns = {};
    for i = 1:length(str) - shingle_size+1
        shingle = str(i:i+shingle_size-1);
        shinglesAns{i} = shingle;
    end

    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

    distJ = ones(1, size(rest,1));
    h = waitbar(0,'Calculating');
    for i=1:size(rest, 1)
        waitbar(i/K, h);
        distJ(i) = sum(MinHashSig(i,:) ~= MinHashString)/K;
    end
    delete(h);

    flag = false;
    temp = 5;
    for i = 1:temp
        [val, pos] = min(distJ);
        if (val <= threshold)
            if ~ismissing(rest{pos, 6})
                flag = true;
                fprintf('Nome: %s; Localidade: %s; Prato(s) recomendado(s):
%s.\t(%f)\n', rest{pos, 2}, rest{pos, 3}, rest{pos, 6}, val);
            end
        end
        distJ(pos) = 1;
    end

    if (~flag)
        fprintf('No dishes found.\n');
    end
end
```

Nesta função era pedido uma string ao utilizador e tínhamos de encontrar quais restaurantes tinham pratos mais similares à string introduzida. Usámos o mesmo método que na função *similarRestaurant* para calcular as distâncias de Jaccard entre os *shingles* da *string* introduzida e os *shingles* dos pratos recomendados. Depois disso vemos qual o valor mínimo das distâncias de Jaccard e enquanto esse valor mínimo for menor ou igual ao *threshold* definido, dizemos que não foram encontrados pratos. De modo a manter a ordem do filme mais similar, damos o valor de 1 à distância de Jaccard do restaurante que foi impresso fazendo com que esse restaurante nunca mais seja selecionado como o mínimo.

selectRestaurant

```
function option = selectRestaurant(rest, file, userIDs, id, M, k4, restAval)
    restaurentes = zeros(1, length(userIDs));
    rlen = 1;

    for i = 1:length(userIDs)
        if userIDs(i) == id
            restaurentes(rlen) = file(i, 2);
            rlen = 1 + rlen;
        end
    end

    restIDs = rest(:, 1)';
    restName = rest(:, 2)';
    restLocal = rest(:, 3)';

    for i = 1:length(restIDs)
        if ismember(restIDs{i}, restaurentes)
            fprintf("ID: %d; Nome: %s; Localidade: %s.\n", restIDs{i},
restName{i}, restLocal{i})
        end
    end

    restIds_int = zeros(length(restIDs));
    for i = 1:length(restIDs)
        if ismember(restIDs{i}, restaurentes)
            restIds_int(i) = restIDs{i};
        end
    end

    option = input('Select an ID: ');
    if ismember(option,restIds_int)
        fprintf('Valid')
    else
        clc;
        fprintf('Invalid option.')
        return
    end

    clc;
    nm = length(M);
```

```
    J = zeros(nm,nm);

    for n1 = 1:nm
        for n2 = n1 + 1:nm
            J(n1, n2) = sum(M(:, n1) == M(:, n2)) / k4;
            J(n2, n1) = J(n1, n2);
        end
    end

    n1 = option;

    topSimilarities = [];
    for n2 = 1:nm
        topSimilarities = [topSimilarities; n1, n2, J(n1, n2),
restAval(n2,2)];
    end

    sortedSimilarities = sortrows(topSimilarities, [-3, -4]);
    Top3IDs = [sortedSimilarities(1,2) sortedSimilarities(2,2)
sortedSimilarities(3,2)];

    for i = 1:length(Top3IDs)
        id4 = Top3IDs(i);
        fprintf("ID: %d; Nome: %s; Localidade: %s.\n", restIDs{id4},
restName{id4}, restLocal{id4})
    end
end
```

Esta função permite ao utilizador selecionar um restaurante de uma lista de restaurantes que ele já avaliou e em seguida, a função calcula a similaridade entre os restaurantes avaliados pelo utilizador e todos ou outros e recomenda os três mais similares.

A parte inicial é semelhante à primeira opção, a função *restaurantYouEvaluated*, posteriormente calcula a matriz de similaridade entre todos os restaurantes.

A matriz *topSimilarities* guarda em cada linha, o ID do restaurante selecionado, um ID de um outro restaurante, a similaridade entre estes 2 IDs e a avaliação média do restaurante do segundo ID feita pelos utilizadores.

Depois de feita a matriz, ela é ordenada principalmente pela maior similaridade entre os 2 Ids, e para similaridades iguais durante a comparação, a matriz ordena esses valores pelo maior valor da avaliação média do restaurante.

No final da ordenação dos restaurantes, são selecionados os 3 melhores restaurantes de acordo com a sua similaridade e avaliação média, e são mostrados pela aplicação.

opção 5

```
tmp = input('Insert User ID: ');
if ~ismember(tmp, userIDs)
    fprintf('Invalid User ID.');
    pause(2);
```

```
                else
                    id = tmp;
                    number = countFBC(data.B,id,data.k);
                    fprintf('Number of evaluations made by User ID %d: %d
\n', id, number);
                    fprintf('\nPress the any key to go back.');
                    pause;
```

Nesta opção não usámos uma função devido à sua simplicidade. Tem como objetivo mostrar o número de avaliações feitas pelo utilizador inserido.

É pedido ao utilizador que insira o ID de um utilizador sendo posteriormente feita, a sua validação. Caso seja aprovado será chamada a função countFBC com os argumentos *data.B*, *id* e *data.k*. Para cada função de hash (*k*), ela calcula um valor de hash para o elemento e mapeia o valor de hash para uma posição no FBC (*B*). No final de avaliar todas as funções de hash, é selecionado o menor valor hash obtidos

Conclusão

Para o desenvolvimento desta aplicação tivemos de usar diversos conceitos explorados nas aulas práticas, também foi necessária a sua adaptação de modo a desenvolver novas funções. Optámos por valores de *k* entre 100 e 150 nas MinHashs visto que este valor fará obter valores de similaridade bastante próximo do real e k = 3 no Filtro de Bloom de contagem pois é um k ótimo de acordo com os valores definidos por m (2426) e n (10000). A aplicação assim poderá fornecer dados com mais precisão.