Relatório - MPEI PL4

Universidade de Aveiro

Diogo Branco Alves da Silva, Bruno Ferreira Gomes

Relatório - MPEI PL4

LECI

Universidade de Aveiro

Diogo Branco Alves da Silva, Bruno Ferreira Gomes
(104341) diogobranco.as@ua.pt, (103320) brunofgomes@ua.pt

23/12/2023

Índice

Capítulo 1

Introdução

Para este trabalho foram nos entregues três ficheiros denominados por *travels1.data*, *tourists1.txt* e *countries_ info.csv* para desenvolver em Matlab funcionalidades de um sistema online de apoio a viagens.

A aplicação deve começar por pedir o ID do utilizador (turista) que se torna o utilizador atual, uma vez o ID estando validado a aplicação irá permitir a seleção de uma das seguintes 5 opções:

1. Listar todos os países que o utilizador visitou.
2. Listar o conjunto dos países avaliados pelos 2 utilizadores mais "similares"ao utilizador atual.
3. Sugerir países a visitar.
4. Sugerir turistas com interesses semelhantes
5. Estimar o total de visitas aos países visitados pelo usuário.

Ao longo deste relatório serão justificadas as opções tomadas pelos alunos na implementação dos diversos métodos probabilísticos recorridos para a resolução dos exercícios.

Capítulo 2

Interface, Main e Dados

2.1 main.m

O *main.m* é o centro do programa. É onde todas as funções do código são chamadas, seja para executar as funcionalidades principais da aplicação ou para lidar com a interface.

2.2 interfaceSelecao.m

Este script tem a função de gerenciar a interação do usuário com o menu da aplicação. Ele mostra as opções disponíveis, recebe as entradas do usuário e valida esses valores antes de prosseguir.

2.3 Suporte.m

Este script lê os três ficheiros com dados e guarda em estruturas de dados as matrizes de assinaturas para as várias opções, os Counting Bloom filters e várias variáveis necessárias para as vária opções. Iremos usar excertos do Suporte.m para explicar o código das opções mais tarde em detalhe.

Capítulo 3

Opção 1

3.1 getCountriesVisited.m

```
1  function visitedCountries = getCountriesVisited(userID
       )
2      Suporte = load("Suporte.mat");
3      travels = Suporte.travels;
4      countries = Suporte.countries;
5      tourists = Suporte.tourists;
6
7      m = travels(userID == travels(:,1), :); % m has 4
           columns, column 1 has the tourist ID, column 2
           has the country ID
8      disp(m);
9
10     visitedCountries = {};
11
12     for i = 1:size(m, 1) % iterates through each row
           of m to find the country name
13         countryID = m(i,2);
14         countryName = countries{countryID, 1};
15         visitedCountries = [visitedCountries,
               countryName]; %add country name to list
16     end
17
18     tourist = tourists(userID == cell2mat(tourists
           (:,1)), :);
19     userName = [tourist{1,2}, ' ', tourist{1,3}];  %
           contatenate first and last name of the user
20
```

```
21      visitedCountries = unique(visitedCountries); %
            remove repeated countries
22
23      fprintf('Countries visited by %s:\n', userName);
24      for i = 1:length(visitedCountries)
25          fprintf('->%s\n', visitedCountries{i});
26      end
27      fprintf("\n");
28
29  end
```

Esta função é chamada pela main quando o usuário seleciona a opção 1 do menu e tem como objetivo listar todos os países visitados pelo usuário.

Após receber o ID do usuário como entrada esta função carrega um ficheiro chamado *Suporte* que contém três variáveis: travels, countries e tourists.

- *travels* é uma matriz onde a coluna 1 contém os IDs dos turistas e a coluna 2 contém os IDs dos países visitados por esses turistas.
- *countries* é uma matriz de células onde a coluna 1 contém os nomes dos países.
- *tourists* é uma matriz de células onde a coluna 1 contém os IDs dos turistas, a coluna 2 contém os primeiros nomes dos turistas e a coluna 3 contém os apelidos dos turistas.

Em seguida o código filtra a matriz *travels* para obter apenas a informação do usuário em questão. Para cada país visitado, o código procura o nome do país na matriz *countries* e adiciona-o à lista *visitedCountries*, países repetidos serão removidos usando a função *unique*. Por fim, é procurado o primeiro nome e apelido do usuário na matriz *tourists* e são finalmente impressos os países visitados por este.

Capítulo 4

Opção 2

4.1 MinHashOp2

```
1
2  K = 100;
3  MinHashOp2 = inf(numberOfTourists,K);
4  h = waitbar(0,'Calculating MinHash Op2');
5  for i = 1:numberOfTourists
6      waitbar(i/numberOfTourists, h);
7      setOfCountries = countriesByTourist{i};
8
9      for j = 1:length(setOfCountries)
10         chave = char(setOfCountries(j));
11         hash = zeros(1,K);
12         for z = 1:K
13             chave = [chave num2str(z)];
14             hash(z) = DJB31MA(chave,127);
15         end
16         MinHashOp2(i,:) = min([MinHashOp2(i,:),hash]);
17     end
18 end
19 delete(h)
```

Esta secção de código que está dentro do Suporte.m gera uma matriz de assinatura para cada um dos turistas, o **K** é o número de Hash Functions que o algoritmo MinHash usa, quanto maior for o número K, maior será a precisão dos resultados mas o tempo de processamento será maior, como mais tarde vamos iterar para escolher os utilizadores mais parecidos, decidimos que 100 era um número bom visto que nos testes demorava cerca de 1 minuto.

4.2 findSimilarUsersAndVisitedCountries

```
function findSimilarUsersAndVisitedCountries(userID)

% Load the MinHash signatures
load('Suporte.mat', 'MinHashOp2', 'touristsIds', '
    touristMap','countriesByTourist','countries');

% Find the index of the given user ID in the
    touristsIds array
userIndex = find(touristsIds == userID);

% Calculate the Jaccard similarity between the given
    user and all other users
jaccardSimilarities = sum(MinHashOp2 == MinHashOp2(
    userIndex, :), 2) / size(MinHashOp2, 2);

% Exclude the given user from the results
jaccardSimilarities(userIndex) = 0;

% Find the indices of the two users with the highest
    Jaccard similarity
[~, closestUserIndices] = maxk(jaccardSimilarities, 2)
    ;

% Get the IDs of the closest users
closestUserIDs = touristsIds(closestUserIndices);
fprintf('The 2 most similair users are: \n')
% Get the names of the closest users
for i = 1:2
    id = closestUserIDs(i);
    name = touristMap(id);
    fprintf('User %d: %s %s \n', id, name{1}, name{2})
        ;
end

fprintf('\n')
fprintf('List of countries visited\n')
fprintf('\n')
% Get the countries visited by the closest users
for i = 1:2
    id = closestUserIDs(i);
    userIndex = find(touristsIds == id);
```

```
35      visitedCountriesIDs = countriesByTourist{userIndex
            };
36      visitedCountriesNames = countries(
            visitedCountriesIDs, 1);
37      for j = 1:length(visitedCountriesNames)
38          fprintf('%s\n', visitedCountriesNames{j});
39      end
40  end
41  end
```

Ao escolher a opção 2 na main, será corrida esta função que usa o ID escolhido e o MinHashOp2 para devolver os 2 utilizadores mais semelhantes e depois mostra a lista de países visitados por esses 2 utilizadores.

Capítulo 5

Opção 3

5.1 MinHashOp3

```
1
2  shingle_size = 11; % Tamanho do shingle
3  K = 50;           % Numero de funcoes de dispersao
4  numberOfCountries = size(countries, 1);
5  MinHashOp3 = inf(numberOfCountries,K);
6
7  h = waitbar(0,'Calculating MinHash Op3');
8  for i = 1:numberOfCountries
9      waitbar(i/numberOfCountries,h);
10
11     % Get the description of the country
12     descriptions = countries{i, 2:end};
13     for j = 1:length(descriptions)
14         description = lower(countries{i, 2});
15     end
16     shingles = {};
17     % Criacao de shingles para a descricao
18     for j = 1:length(description) - shingle_size + 1
19         shingle = description(j:j+shingle_size-1);
20         shingles{j} = shingle;
21     end
22
23     for j = 1:length(shingles)
24         chave = char(shingles(j));
25         hash = zeros(1,K);
26         for z = 1:K
27             chave = [chave num2str(z)];
28             hash(z) = DJB31MA(chave,127);
```

```
        end
        % Valor minimo da hash para este shingle
        MinHashOp3(i,:) = min([MinHashOp3(i,:);hash]);
    end
end
delete(h);
```

O shingle_size determina o tamanho das substrings de texto que serão criadas a partir da descrição dos vários países, nos nossos testes, como vamos ter descrições de texto muito grandes, decidimos usar um shingle_size de 10 pois se usarmos um tamanho menor o nosso algoritmo será mais sensível para similaridades não importantes.

Não notamos muitas diferenças quanto ao tempo de processamento com a mudança do shingle_size entre 7 e 10 no entanto o numéro de HashFunctions criadas **K** afetou muito o tempo de processamento, decidimos então diminuir o numero sabendo que os resultados serão menos precisos.

MinHashOp3 guarda os países e o seu grau de similaridade com outros países tendo em relação os conteúdos da sua descrição

5.2 findMostSimilarCountries

```

function mostSimilarCountries =
    findMostSimilarCountries(userID)
% Load the necessary data
load('Suporte.mat','countriesByTourist','MinHashOp3','
    countries','touristsIds');

% Find the index of the given user ID in the
    touristsIds array
userIndex = find(touristsIds == userID);

% Get the unique countries visited by the user
visitedCountriesIDs = unique(countriesByTourist{
    userIndex});

% Create a cell array for the most similar countries
mostSimilarCountries = cell(length(visitedCountriesIDs
    ), 2);

% Loop over each visited country
```

```
for i = 1:length(visitedCountriesIDs)
    visitedCountryID = visitedCountriesIDs(i);

    % Get the MinHash signature of the visited country
    visitedCountrySignature = MinHashOp3(
        visitedCountryID, :);

    % Get the IDs and MinHash signatures of the
        unvisited countries
    unvisitedCountriesIDs = setdiff(1:size(countries,
        1), visitedCountryID);
    unvisitedCountriesSignatures = MinHashOp3(
        unvisitedCountriesIDs, :);

    % Calculate the Jaccard similarity between the
        visited and unvisited countries
    jaccardSimilarities = sum(visitedCountrySignature
        == unvisitedCountriesSignatures, 2) / size(
        visitedCountrySignature, 2);

    % Create a cell array of unvisited countries and
        their distances
    unvisitedCountriesAndDistances = cell(length(
        unvisitedCountriesIDs), 2);
    for j = 1:length(unvisitedCountriesIDs)
        unvisitedCountriesAndDistances{j, 1} =
            countries{unvisitedCountriesIDs(j), 1};
        unvisitedCountriesAndDistances{j, 2} =
            jaccardSimilarities(j);
    end

    % Sort the array based on the distances
    unvisitedCountriesAndDistances = sortrows(
        unvisitedCountriesAndDistances, 2);

    % Print the list of unvisited countries and their
        distances
    fprintf('List of unvisited countries and their
        distances:\n');
    for j = 1:size(unvisitedCountriesAndDistances, 1)
        fprintf('%s: %f\n',
            unvisitedCountriesAndDistances{j, 1},
            unvisitedCountriesAndDistances{j, 2});
    end

```

```
45      % Find the unvisited country with the highest
            Jaccard similarity
46      [~, mostSimilarCountryIndex] = max(
            jaccardSimilarities);
47
48      % Add the most similar country to the array
49      mostSimilarCountries{i, 1} = countries{
            visitedCountryID, 1};
50      mostSimilarCountries{i, 2} = countries{
            unvisitedCountriesIDs(mostSimilarCountryIndex),
             1};
51  end
52
53  % Print the list of most similar countries
54  fprintf('List of most similar countries:\n');
55  for i = 1:size(mostSimilarCountries, 1)
56      fprintf('The most similar unvisited country to %s
            is: %s\n', mostSimilarCountries{i, 1},
            mostSimilarCountries{i, 2});
57  end
58  end
```

O findMostSimilarCountries recebendo o UserID e a matriz de Assinatura MinHashOp3 mostra os todos os países não visitados pelo Utilizador e as suas respetivas distâncias e após, mostra para cada país que o Utilizador visitou o país mas parecido que não foi visitado conforme a sua descrição.

Capítulo 6

Opção 4

6.1 MinHashOp4

```
% Number of tourists
numberOfTourists = length(touristsIds);

% Build a list of interests for each tourist
interestsByTourist = cell(numberOfTourists,1);
for i = 1:numberOfTourists
    % Get the interests of each tourist
    lines = find(cell2mat(tourists(:,1)) ==
       touristsIds(i));
    % Store in a cell array
    interestsByTourist{i} = [interestsByTourist{i}
       tourists(lines,4:end)];
end

K = 100;
MinHashOp4 = inf(numberOfTourists,K);
h = waitbar(0,'Calculating MinHash Op4');
for i = 1:numberOfTourists
    waitbar(i/numberOfTourists, h);
    setOfInterests = interestsByTourist{i};

    for j = 1:size(setOfInterests, 2)
        chave = char(setOfInterests(j));
        hash = zeros(1,K);
        for z = 1:K
            chave = [chave num2str(z)];
            hash(z) = DJB31MA(chave,127);
        end
```

```
        MinHashOp4(i,:) = min([MinHashOp4(i,:),hash]);
    end
end
delete(h)
```

O MinHashOp4 gera uma matriz de assinatura para cada um dos turistas com os valores de similaridade dos seus gostos.

6.2 findSimilarTourists

```
function findSimilarTourists(userID)
% Load the variables from the Suporte.mat file
load('Suporte.mat', 'MinHashOp4', 'touristMap', '
    touristsIds', 'numberOfTourists');

% Find the index of the user in the touristsIds array
userIndex = find(touristsIds == userID);

% Get the MinHash signature of the user
userSignature = MinHashOp4(userIndex, :);

% Calculate the Jaccard similarity between the user
    and all other tourists
jaccardSimilarities = zeros(1, numberOfTourists);
for i = 1:numberOfTourists
    jaccardSimilarities(i) = sum(userSignature ==
        MinHashOp4(i, :)) / size(MinHashOp4, 2);
end

% Find the indices of the tourists with the highest
    Jaccard similarity
[~, sortedIndices] = sort(jaccardSimilarities, '
    descend');

% Get the IDs of the most similar tourists
similarTouristsIds = touristsIds(sortedIndices);

% Print the IDs of the most similar tourists
userNames = touristMap(userID);
fprintf('The tourists with the most similar interests
    to %s %s are:\n', userNames{1}, userNames{2});
%10 most similair users based on interests excluding
    the userID
for i = 2:11
```

```
28      touristName = touristMap(similarTouristsIds(i));
29      fprintf('%s %s\n', touristName{1}, touristName{2})
            ;
30  end
31  end
```

Na função findSimilarTourists usamos o userID e o MinHashOp4 para encontrar os turistas com interesses mais parecidos aos do utilizador escolhido.

Capítulo 7

Opção 5

7.1 getVisitCount.m

```
1
2  function totalEstimatedCount = getVisitCount(userID)
3      load('Suporte.mat', 'travels', 'countries', '
           bloomFilter');
4
5      m = travels(userID == travels(:,1), :); % m has 4
           columns, column 1 has the tourist ID, column 2
           has the country ID
6
7      visitedCountries = {};
8
9      for i = 1:size(m, 1) % iterates through each row
           of m to find the country name
10         countryID = m(i,2);
11         countryName = countries{countryID, 1};
12         visitedCountries = [visitedCountries,
               countryName]; %add country name to list
13     end
14
15
16     totalEstimatedCount = 0;
17
18     for i = 1:length(visitedCountries)
19         countryName = visitedCountries{i};
20
21         % add visit to Bloom filter
22         bloomFilter = adicionar_elemento(bloomFilter,
               countryName, 3);
```

```
23
24          % check if visit is in Bloom filter and
                increment total count
25          if pertenca(bloomFilter, countryName, 3)
26              totalEstimatedCount = totalEstimatedCount
                    + 1;
27          end
28      end
29
30      fprintf('%d total visits to countries by current
            user.\n', totalEstimatedCount);
31  end
```

Este script calcula e apresenta o número total estimado de visitas a países por um utilizador específico com base em dados armazenados em estruturas.

- Carrega os dados de viagens, países e um filtro de Bloom a partir do ficheiro 'Suporte.mat';
- Identifica as viagens associadas a um determinado ID de utilizador. Obtém os nomes dos países visitados pelo utilizador a partir do ID do país;
- Utiliza um filtro de Bloom para guardar e contar as visitas aos países;
- Apresenta o número total estimado de visitas à saída.

O código utiliza funções auxiliares, adicionar_elemento() e pertenca(), que permitem a manipulação do filtro de Bloom.

Capítulo 8

Instruções

- Correr o ficheiro "Suporte.m"
- Correr o ficheiro "Main.m"

Capítulo 9

Conclusões

Achamos o trabalho um bocado difícil pois na aula não consguimos colocar a função para as assinaturas Min Hash estarem a functionar, mesmo assim fizemos progresso no trabalho e achamos que está num estado apresentável.