deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Guião de Avaliação 02

Algoritmos Probabilísticos

2023/24

Métodos Probabilísticos para Engenharia Informática

Prof. Carlos Alberto da Costa Bastos - P2

Henrique Ferreira Teixeira - 114588
Rui de Faria Machado - 113765

Dezembro, 2023

Índice

Introdução

O objetivo deste trabalho, realizado no contexto da disciplina de Métodos Probabilísticos para Engenharia Informática, foi o desenvolvimento de uma aplicação em Matlab que oferece funcionalidades de um sistema online para informação e recomendação de restaurantes na região norte de Portugal.

Para este fim, foram fornecidos pelo professor dois conjuntos de dados. O primeiro, um arquivo chamado 'turistas1.data', contém informações detalhadas sobre as avaliações dos restaurantes feitas por turistas, incluindo o ID do turista, o ID do restaurante avaliado e a pontuação atribuída. O segundo, denominado 'restaurantes.txt', fornece uma visão abrangente sobre cada restaurante, incluindo detalhes como ID, nome, localidade, concelho, tipo de cozinha, pratos recomendados e dias de encerramento.

Manipulação dos ficheiros disponibilizados

Tendo em conta que os ficheiros disponibilizados continham uma grande quantidade de informação, eles foram processados de maneira a tornar o seu uso mais acessível e conveniente.

- *turistas.data*

```
turistas_data = load('turistas1.data')
turistas = turistas_data(:,1);
restaurantes = turistas_data(:,2);
score = turistas_data(:,4);
```

- *ficheiros.txt*

```
rest = readcell('restaurantes.txt', 'Delimiter','\t')
restaurantes_nomes = rest(:,2);
restaurantes_nomes_og = rest(:,2);
```

Estas variáveis foram armazenadas em ficheiros ‘.mat’ no script ‘load_data’.

Opção 1

Objetivo

Listar os restaurantes avaliados pelo utilizador atual. Apresentando o seu ID, nome e concelho onde está localizado.

Implementação

1. Procurar e listar os restaurantes visitados pelo utilizador no ficheiro *turistas1.data*
2. Procurar e listar o respectivo ID, nome e concelho no ficheiro *restaurantes.txt*.

Desta forma foi criada a função option1:

```
function visited_restaurantes_id = option1(restaurants_ids, rest)
   restaurants_ids = sort(restaurants_ids);
   fprintf('%3s | %-40s | %s \n', 'ID','Name','County');
   fprintf('%3s | %-40s
|%s\n','---','----------------------------------------','----------');
   visited_restaurantes_id = [];
   for i = 1:length(restaurants_ids)
       for j = 1:length(rest)
           if restaurants_ids(i) == cell2mat(rest(j,1))
               fprintf('%3d | %-40s | %s \n',
restaurants_ids(i),cell2mat(rest(j,2)),cell2mat(rest(j,4)));
               visited_restaurantes_id =
[visited_restaurantes_id,restaurants_ids(i)];
           end
       end
   end
end
```

Esta foi inicializada da seguinte forma: `option1(Set{ID},rest);`

Opção 2

Objetivo

Apresentar o conjunto de restaurantes avaliados pelo utilizador mais “similar” ao utilizador atual.

Implementação

1. Criação da Matriz de Entrada, na qual as linhas correspondem aos restaurantes e as colunas aos turistas. Para tal, foi criada a função *formar_matriz.m*. Usamos duzentas funções de dispersão devido a no guião 4 das aulas práticas ter sido descoberto através de testes que este seria o valor mais adequado.

```
Nu = length(users);
k = 200;
mat = formar_matriz(Set);
```

2. Criação da Matriz de Assinaturas a partir da função *minHash.m*.

```
[a,b,c,pp] = dispersao(k);
assinaturas = minHash(mat,k,a,b,c,pp);
distancias = distancias_j(Nu,k,assinaturas);
distanciasTemp = distancias(ID, :);
distanciasTemp(ID) = Inf;
```

3. Encontrar o valor mínimo (maior similaridade) e respetivo índice

```
[minimo, indiceColuna] = min(distanciasTemp);

% Verificar se o mínimo é zero e encontrar o próximo mínimo
if minimo == 0
      distanciasTemp(indiceColuna) = Inf; % Excluir o zero
      [minimo, indiceColuna] = min(distanciasTemp);
end
```

4. Listar todos os restaurantes que o utilizador mais similar visitou

```
[minimo, indiceColuna] = min(distanciasTemp);

if minimo == 0
      distanciasTemp(indiceColuna) = Inf; % Excluir o zero
      [minimo, indiceColuna] = min(distanciasTemp);
      end

fprintf('List of restaurants visited by the most similar user:\n ');
disp(Set{indiceColuna}');
```

Opção 3

Objetivo

Procurar restaurante através de uma string de input. Apresentando os, no máximo, 5 restaurantes, ordenados por ordem crescente de distância de Jaccard, apenas se esta foi menor ou igual a 0.99.

Implementação

1. Pedir ao utilizador para inserir uma string

```
opt3 = '';
while length(opt3) < 2
   opt3 = input('Write a string: ','s');
end
```

2. Apresentar a lista de restaurantes similares

   A função 'option3' faz o 'hashing' através de shingles. Após uma profunda análise, concluímos que o tamanho de shingle mais adequado é 2. Embora isto tenha um custo na performance, este é tão reduzido que não é considerado relevante. Com um menor tamanho do shingle haverá um maior número destes, e assim, um maior universo de comparação, ou seja, uma maior precisão.

```
shingle_size = 2;
k = 200;
[a,b,c,pp] = dispersao(k);
assinaturas_input = option3(opt3, shingle_size, k,a,b,c,pp);
%tornar a cell com os restaurantes s/ caracteres special
similaridades_total = zeros(1, length(restaurantes_nomes));
for i = 1:length(restaurantes_nomes)
   nome_restaurante = restaurantes_nomes{i};
   % Verificar se o nome do restaurante é longo o suficiente
   if length(nome_restaurante) >= shingle_size
       matriz_restaurantes = option3(nome_restaurante, shingle_size,
k,a,b,c,pp);
       % Verificar se a matriz retornada é não-vazia
       if ~isempty(matriz_restaurantes)
           similaridades_total(i) = sum(assinaturas_input ==
matriz_restaurantes) / k;
       else
           % Lidar com a situação em que a matriz é vazia
           similaridades_total(i) = 0;
       end
   else
       % Lidar com nomes de restaurantes muito curtos
```

```
        similaridades_total(i) = 0;
    end
end
distancias_J = 1-similaridades_total;
% Ordenar as distâncias de Jaccard
[distances_sorted, indices_sorted] = sort(distancias_J);
% Imprimir os 5 restaurantes com menores distâncias
fprintf('The restaurants closest to "%s" are:\n', opt3);
if any(distances_sorted(1:5) > 0.99)
    fprintf('There are no restaurants similiar enough (d <= 0.99 )to
the string entered');
    return
else
    for i = 1:5
        fprintf('%-40s | ID = %-3d\n',
restaurantes_nomes{indices_sorted(i)}, indices_sorted(i));
    end
end
```

Opção 4

Objetivo

Apresentar os 3 restaurantes mais similares ao restaurante escolhido pelo utilizador atual.

Implementação

1. Listar informação relativa aos restaurantes visitados pelo utilizador atual. Isto inclui ID, nome e concelho

```
fprintf('Restaurants evaluated by the user:\n')
visited_res=option1(Set{ID},rest);
```

2. Pedir ao utilizador que escolha um dos restaurantes que avaliou.

```
if isempty(visited_res)
   fprintf('User %d is yet to review a restaurant!', ID);
   break;
else
   idRestauranteEscolhido = input('Choose the restaurant ID: ');
   while ~any(visited_res == idRestauranteEscolhido)
       fprintf('Erro: The restaurant chosen has not been evaluated by
the user.\n');
       idRestauranteEscolhido = input('Choose the restaurant ID: ');
   end
end
```

3. Apresentar os 3 restaurantes mais similares ao escolhido
    a. Primeiro nível de comparação: Baseado na similaridade do conjunto de campos associados

```
rest_chopped = rest(:,3:end);
k = 200;
[a,b,c,pp]=dispersao(k);
shingle_size = 2;
count = 0;
similaridade_rest = zeros(1,length(rest_chopped));
for j = 1:width(rest_chopped)
   componente = rest_chopped{idRestauranteEscolhido,j};
   if all(~ismissing(componente)) && all(length(componente) >=
shingle_size)
       assinat_escol = option3(componente,shingle_size,k,a,b,c,pp);
       count = count + 1;
       for i = 1:length(rest_chopped)
```

```
                word = rest_chopped{i,j};
                lio = length(word);
                if all(~ismissing(word)) && all(length(word) >=
shingle_size)
                    hash_rest = option3(word,shingle_size,k,a,b,c,pp);
                    similaridade_rest(i) = similaridade_rest(i) +
sum(assinat_escol == hash_rest) / k;
                end
            end
        end
    end
    if count > 0
        similaridade_media = similaridade_rest / count;
    else
        similaridade_media = 0;
    end
    % Ordena as similaridades em ordem decrescente
    [rest_sim_sorted, rest_indice_sorted] = sort(similaridade_media,
'descend');
```

b. Segundo nível de comparação: Baseado no valor médio de avaliação dos restaurantes

i. Cálculo das médias de avaliações dos restaurantes

```
[Set_rest, index] = create_struct(turistas_data(:,[2,4]));
average_review = zeros(1,length(Set_rest));
for i = 1:length(Set_rest)
    average_review(i) = sum(Set_rest{i})/length(Set_rest{i});
end
```

ii. Ordenação dos restaurantes por similaridade e identificação dos grupos de desempate

```
gruposDesempate = cell(1, 3);
indiceGrupo = 1;
% Percorre todos os restaurantes ordenados por similaridade
for indiceAtual = 1:length(rest_sim_sorted)
    if indiceGrupo > 3
        % Interrompe a criação de novos grupos após três grupos
        break;
    end
    similaridadeAtual = rest_sim_sorted(indiceAtual);
    grupoAtual = similaridadeAtual;
    % Percorre enquanto houver restaurantes com a mesma
similaridade
    while indiceAtual + 1 <= length(rest_sim_sorted) &&
rest_sim_sorted(indiceAtual + 1) == similaridadeAtual
        indiceAtual = indiceAtual + 1;
        grupoAtual = [grupoAtual, rest_sim_sorted(indiceAtual)];
    end
    % Adiciona o grupo atual à lista de grupos de desempate
```

```
            gruposDesempate{indiceGrupo} = grupoAtual;
            indiceGrupo = indiceGrupo + 1;
            indiceAtual = indiceAtual + 1;
        end
```

iii. Desempate dentro dos grupos com as médias de avaliações e reconstrução da lista ordenada e seleção dos 3 restaurantes mais similares

```
        % Aplicar o desempate por média de avaliações
        podio = zeros(1, 3);
        for i = 1:length(gruposDesempate)
            grupo = gruposDesempate{i};
            if length(grupo) > 1 % Se houver mais de um restaurante no
grupo
                indicesGrupo = find(ismember(similaridade_media, grupo));
                avaliacoes = average_review(indicesGrupo);
                [~, ordem] = sort(avaliacoes, 'descend');
                indicesOrdenados = indicesGrupo(ordem);
                lugaresLivres = find(podio == 0,
min(length(indicesOrdenados), 3));
                podio(lugaresLivres) =
indicesOrdenados(1:length(lugaresLivres));
            else
                % Caso haja apenas um restaurante no grupo
                index = find(similaridade_media == grupo);
                if length(index) == 1 % index seja um único valor
                    if any(podio == 0) % Se houver lugar livre no pódio
                        primeiroLivre = find(podio == 0, 1);
                        podio(primeiroLivre) = index;
                    end
                end
            end
        end
```

Opção 5

Objetivo

Apresentar uma estimativa do número de avaliações para um restaurante escolhido pelo utilizador atual.

Implementação

1. Pedir o ID do restaurante e respetiva validação

```
IDRes = input("Restaurant ID: ");
while IDRes < 0 ||  IDRes > 213
   fprintf("Invalid ID")
   IDRes = input("Restaurant ID: ");
end
```

2. Criação da função *Initialize.m* para criar o Filtro de Bloom. Esta foi inicializada com os seguintes parâmetros:

   - $m = length(restaurantes) = 213$
   - $p = 0.01 \; -> \; probabilidade\ máxima\ aceitável\ de\ um\ obter\ um\ falso\ positivo$
   - $n = -\frac{m \times ln(p)}{(ln(2))^2} \simeq 2042$
   - $k = \frac{n \times ln(2)}{m} \simeq 7$

```
BloomFilter = Initialize(m , n , k);
```

3. Percorrer o ficheiro *turistas.data* e inserir todos os restaurantes (com repetição) avaliados no filtro. Para tal, foi criada a função *Insert.m*.

```
for i = 1:size(turistas)
   idRestaurante = restaurantes(i);
   BloomFilter = Insert(BloomFilter, idRestaurante);
end
```

4. Estimar o número de avaliações de um restaurante inserido pelo utilizador atual usando a função *EstimateReviews.m*.

```
estimativa = EstimateReviews(BloomFilter, IDRes);
fprintf('The estimated number of evaluations for restaurant\n %s | ID
= %d is %d\n', restaurantes_nomes_og{IDRes}, IDRes, estimativa);
```

Conclusão

Este guião de avaliação permitiu-nos aplicar diversos conceitos teóricos numa aplicação prática, em Matlab. As várias opções implementadas ilustram a utilidade de métodos probabilísticos em situações reais.

O menu de navegação e a opção 6, embora não detalhados aqui por sua natureza trivial, foram também implementados com sucesso, completando a funcionalidade da aplicação.

Este trabalho reforçou as nossas habilidades em programação e análise de dados, essenciais para a engenharia informática. O projeto foi concluído com sucesso, respeitando as boas práticas de programação e demonstrando a eficácia da nossa abordagem metodológica.