# MPEI - PL4

Afonso Baixo, Paulo Macedo

Universidade de Aveiro

# MPEI - PL4

Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Afonso Baixo - afonso.baixo@ua.pt

Paulo Macedo - paulomacedo@ua.pt

21/12/2023

# Conteúdo

**1 Introdução** **1**

**2 Data Structure** **2**
2.1 Bloom Filter . . . . . 5
2.2 Minhash e Shingles . . . . . 6
2.2.1 Discussão . . . . . 7

**3 Aplicação** **8**
3.1 Menu . . . . . 8
3.2 opcao1 . . . . . 10
3.3 opcao2 . . . . . 10
3.4 opcao3 . . . . . 12
3.5 opcao4 . . . . . 13
3.6 opcao5 . . . . . 16

**4 Notas Finais** **18**

# Lista de Figuras

2.1 Valores teóricos para um filtro de 16000 posições e m=2426 . . . 6

# Capítulo 1

# Introdução

Neste projeto em Matlab, vamos criar uma aplicação que visa oferecer informações sobre restaurantes e sugerir opções com base nas avaliações dos utilizadores. Utilizaremos dados provenientes de dois ficheiros principais: o 'utilizadores.data', onde encontramos as avaliações dos utilizadores para diferentes restaurantes, e o 'restaurantes.txt', que contém informações detalhadas sobre cada restaurante, como nome, localidade, tipo de cozinha e outros detalhes relevantes.

No 'utilizadores.data', os utilizadores são identificados por um ID e associam avaliações a diferentes restaurantes através de outro ID. As avaliações estão representadas por scores, permitindo-nos entender as preferências de cada utilizador em relação aos restaurantes visitados.

Por sua vez, o 'restaurantes.txt' contém informações detalhadas sobre os restaurantes, como o seu nome, localização, tipo de cozinha, pratos recomendados e dias de encerramento.

O nosso objetivo é criar uma aplicação em Matlab que utilize estes dados para oferecer aos utilizadores recomendações personalizadas de restaurantes com base nas avaliações existentes e nas características de cada estabelecimento. Ao processar e analisar essas informações, a aplicação será capaz de sugerir opções de restaurantes que possam corresponder aos gostos e preferências individuais de cada utilizador.

Para isso foram inicialmente desenvolvidos 2 scripts Matlab

- 1 - Script desenvolvido para inicializar a estrutura de dados necessários à implementação da aplicação, incluindo todas as matrizes de suporte necessárias às funções desenvolvidas pela aplicação principal.
- 2 - Script desenvolvido para ler do disco todos os dados previamente guardados pelo primeiro script e para a implementação de todas as interações com o utilizador.

# Capítulo 2

# Data Structure

A função `setDataStruct` é responsável por inicializar a estrutura de dados necessária para o funcionamento do sistema de recomendação de restaurantes. Esta função lê os dados dos restaurantes e dos utilizadores de ficheiros externos, realiza processamento para criar estruturas de dados e inicializa um filtro de Bloom.

O processo inicia com a leitura dos dados dos restaurantes a partir do ficheiro 'restaurantes.txt' e os dados dos utilizadores do ficheiro 'utilizadores.data'. São carregadas as colunas relevantes para a aplicação, que contêm informações sobre os IDs dos utilizadores, IDs dos restaurantes avaliados por esses utilizadores e as avaliações atribuídas.

Os IDs únicos dos utilizadores são identificados e armazenados no array `usersID`. Além disso é criado uma estrutura `evalByUser` que contém uma lista de todos os restaurantes avaliados por cada utilizador e uma outra estrutura `avgRating` que contém os pares restaurante e média de avaliação associado ao mesmo.

```
function setDataStruct()
tic
h = waitbar(0, 'Initializing...');

% Reading the restaurant data
waitbar(0.05, h, 'Reading␣restaurant␣data...');
restaurants = readcell('restaurantes.txt', 'Delimiter', '\t');

% Reading user data
waitbar(0.1, h, 'Reading␣user␣data...');
user_data = load('utilizadores.data');
user_data = user_data(:, [1, 2, 4]);

% Get users ID
waitbar(0.15, h, 'Processing␣user␣IDs...');
```

```
usersID = unique(user_data(:,1));

n = 16e3;
hashFuncBloomFilter = 5;
waitbar(0.2, h, 'Creating␣Bloom␣Filter...');
bloomFilter=createBloomFilter(n, hashFuncBloomFilter);

for i=1:length(user_data)
    bloomFilter = insertElement(bloomFilter,user_data(i, 1));
end

waitbar(0.3, h, 'Creating␣restaurant␣sets...');
evalByUser = createRestaurantSets(user_data, usersID);

hashFuncMinHash = 200;
Nu = length(usersID);

% Opcao3 Minhash Signatures
waitbar(0.4, h, 'Calculating␣signatures␣(Evaluations)...');
signatures = inf(Nu, hashFuncMinHash);
for i = 1:Nu
    conjunto = evalByUser{i};
    for j = 1:length(conjunto)
        chave = char(conjunto(j));
        hash = zeros(1,hashFuncMinHash);
        for x = 1:hashFuncMinHash
            chave = [chave num2str(x)];
            hash(x) = DJB31MA(chave,127);
        end
        signatures(i,:) = min([signatures(i,:); hash]);
    end
    waitbar(0.4 + 0.15 * (i / Nu), h);
end

shingle_size=3;
shingles_k = 150;
shinglesSignatures = inf(length(restaurants), shingles_k);

waitbar(0.55, h, 'Calculating␣shingle␣signatures␣(Dish)␣...');
for i = 1:length(restaurants)
    conjunto = lower(restaurants{i,6});
    shingles = {};
    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, shingles_k);
        for x = 1:shingles_k
            chave = [chave num2str(x)];
            hash(x) = DJB31MA(chave,127);
        end
        shinglesSignatures(i, :) = min([shinglesSignatures(i, :);
hash]);  % Valor minimo da hash para este shingle
    end
    waitbar(0.55 + 0.15 * (i / length(restaurants)), h);
end

op4shingle_size=3;
op4shingles_k = 150;
op4shinglesSignatures = inf(length(restaurants), shingles_k);

waitbar(0.7, h, 'Calculating␣shingle␣signatures␣(Dish␣&␣Cuisine)␣
...');
for i = 1:length(restaurants)
    tipoCozinha = restaurants{i,5};
    pratoTipico = restaurants{i,6};
    if(ismissing(tipoCozinha))
        tipoCozinha = '';
    end
    if(ismissing(pratoTipico))
        pratoTipico = '';
    end
    conjunto = lower([tipoCozinha pratoTipico]);
    shingles = {};
    for j = 1 : length(conjunto) - op4shingle_size + 1
        shingle = conjunto(j: j + op4shingle_size - 1);
        shingles{j} = shingle;
    end

    for j = 1:length(shingles)
        chave = char(shingles(j));
        hash = zeros(1, op4shingles_k);
        for x = 1:op4shingles_k
            chave = [chave num2str(x)];
            hash(x) = DJB31MA(chave,127);
        end
        op4shinglesSignatures(i, :) = min([op4shinglesSignatures(i,
:); hash]);  % Valor minimo da hash para este shingle
    end
```

```
        waitbar(0.7 + 0.15 * (i / length(restaurants)), h);
    end

    avgRating = zeros(length(restaurants), 2);
    waitbar(0.85, h, 'Calculating restaurants average score ...');
    for id = 1:length(restaurants)
        evaluatedRows = find(user_data(:,2) == id);
        avg_rating = sum(user_data(evaluatedRows,3)) / length(
    evaluatedRows);
        avgRating(id,1) = id;
        avgRating(id,2) = avg_rating;
        waitbar(0.85 + 0.1 * (id / length(restaurants)), h);
    end

    % Saving data
    waitbar(0.95, h, 'Saving data...');
    save setDataStruct user_data restaurants bloomFilter evalByUser
    signatures usersID hashFuncMinHash shinglesSignatures
    op4shinglesSignatures shingle_size shingles_k op4shingles_k
    op4shingle_size avgRating

    % Close the waitbar
    completionMessage = sprintf('Complete! Took %d seconds', toc);
    waitbar(1, h, completionMessage);
    pause(2);
    close(h);
end
```

## 2.1 Bloom Filter

O filtro de Bloom serve para otimizar a busca de elementos específicos nos dados dos utilizadores. É definido um tamanho para o filtro e um número de funções hash para as operações do filtro.

Após criar o filtro de Bloom, a função itera por todos os dados dos utilizadores e insere esses elementos no filtro, preparando-o para consultas rápidas e eficientes durante a execução da aplicação.

Finalmente, a função gera conjuntos de restaurantes avaliados por cada utilizador, preparando assim a estrutura necessária para a realização de recomendações personalizadas com base nas avaliações dos utilizadores.

Para calcular o número ótimo de funções de dispersão foi feita uma avaliação com estatísticas entre a probabilidade de falsos positivos para cada número de funções de dispersão.

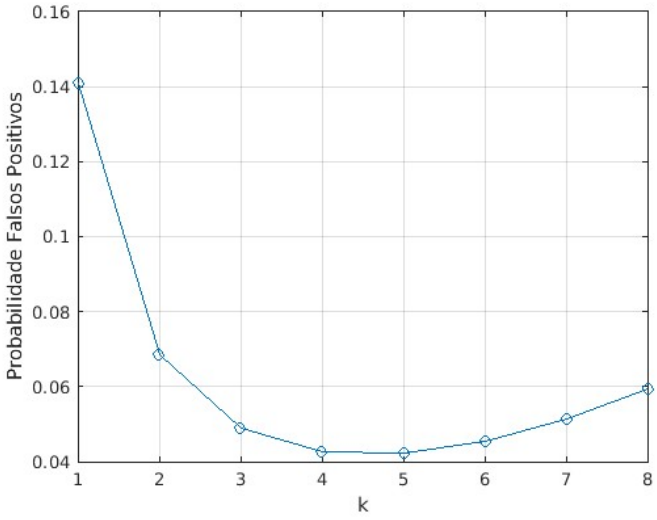


Figura 2.1: Valores teóricos para um filtro de 16000 posições e m=2426

Tendo em conta a figura anterior é possível verificar que o número ótimo de funções de dispersão é `5`.

## 2.2 Minhash e Shingles

No trecho de código de implementação associado, estão presentes operações cruciais para calcular as assinaturas `MinHash` e os `Shingles`, preparando os dados para posterior análise e comparação de similaridade.

Inicialmente, é definido o parâmetro `hashFuncMinHash` que foi decidido através de vários testes com diferentes valores até encontrar o valor que nos dá um resultado mais aproximado daquele que seria obtido através do cálculo clássico da distância de Jaccard mantendo a maior eficiência e rapidez possível. É então iniciado o cálculo das assinaturas `MinHash` para os conjuntos de restaurantes avaliados por cada utilizador. Isso é feito iterando por cada utilizador e os seus conjuntos de restaurantes avaliados.

Para cada restaurante no conjunto, é gerado um conjunto de hashes, utilizando a função `DJB31MA`. As assinaturas `MinHash` são calculadas tomando o valor mínimo de hash para cada conjunto e armazenadas na matriz `signatures`, representando as assinaturas `MinHash` para cada utilizador.

Em seguida, para auxílio da opcao3, o código calcula os shingles para os pratos recomendados pelos restaurantes. Para cada restaurante, a *string* do prato recomendado é dividido em shingles de tamanho específico (`shingle_size`). É então gerado um conjunto de hash para cada shingle, e as assinaturas dos shingles são armazenadas na matriz `shinglesSignatures`, representando os shingles para cada restaurante. Este procedimento é repetido, para auxílio da

opcao4, com o diferente pormenor de que os shingles são calculados tendo em conta ambos, o prato típico e o tipo de cozinha do restaurante.

### 2.2.1 Discussão

#### Número Ótimo de Funções de Dispersão para o Filtro de Bloom

Determinar o número ideal de funções de dispersão para um filtro de Bloom geralmente está relacionado ao tamanho esperado do conjunto de dados e ao nível de tolerância desejado para falsos positivos. Essa otimização tem como objetivo minimizar o número de colisões (falsos positivos) no filtro enquanto mantém um tamanho viável para o filtro.

#### Cálculo dos Shingles

O cálculo de shingles envolve a divisão de um texto (como o nome de um prato recomendado) em subsequências de caracteres de tamanho fixo, conhecidos como shingles. Esse processo é relevante para o sistema de recomendação de restaurantes por algumas razões:

1. **Compactação de Informação**: Dividir o texto em shingles de tamanho específico permite representar o texto de maneira mais compacta e estruturada, facilitando a comparação e análise subsequente.

2. **Comparação de Similaridade**: A representação em shingles permite calcular a similaridade entre diferentes textos. No contexto do sistema de recomendação de restaurantes, isso possibilita identificar pratos ou recomendações de restaurantes semelhantes com base nas sequências de shingles comuns.

3. **Personalização das Recomendações**: Ao analisar os shingles de pratos recomendados por um utilizador e compará-los aos shingles de outros restaurantes, o sistema pode oferecer recomendações mais personalizadas e relevantes com base nas preferências do utilizador.

# Capítulo 3

# Aplicação

## 3.1 Menu

O menu é exibido com uma série de opções numeradas, cada uma correspondendo a uma funcionalidade específica da aplicação:

- Listar os restaurantes avaliados pelo utilizador.
- Encontrar o conjunto de restaurantes avaliados pelo utilizador mais similar.
- Pesquisar por um prato especial nos restaurantes.
- Encontrar os restaurantes mais similares com base nas avaliações dos utilizadores.
- Estimar o número de avaliações por um determinado utilizador.
- Sair da aplicação.

```
clear;

% Check if setDataStruct.mat exists
if exist('setDataStruct.mat', 'file') == 2
    load setDataStruct
else
    setDataStruct()
    load setDataStruct
end

% Get User ID
userID = input('Insert User ID (1 to ??): ');
```

```
% Loop until the user chooses to exit
while true
    % Display menu options
    disp('1 - Restaurants evaluated by you');
    disp('2 - Set of restaurants evaluated by the most similar user');
    disp('3 - Search special dish');
    disp('4 - Find most similar restaurants');
    disp('5 - Estimate the number of evaluations by each tourist');
    disp('6 - Exit');
    choice = input('Select choice: ');

    switch choice
        case 1
            % List Restaurants Evaluated by User
            opcao1(userID, user_data, restaurants);

        case 2
            % Find Most Similar User
            opcao2(userID, user_data, hashFuncMinHash, usersID,
   signatures, restaurants);

        case 3
            % Search Special Dish
            opcao3(restaurants, shinglesSignatures, shingle_size,
   shingles_k);

        case 4
            % Find Most Similar Restaurants
            opcao4(userID, user_data, restaurants,
   op4shinglesSignatures, op4shingle_size, op4shingles_k, avgRating);

        case 5
            % Estimate Evaluations by a User
            opcao5(user_data, bloomFilter);

        case 6
            % Exit Application
            fprintf('Exiting application.\n\n');
            break;

        otherwise
            fprintf('Invalid choice. Please select a valid option.\n\n'
   );
    end
end
```

Dependendo da opção escolhida pelo utilizador, o código chama funções correspondentes (`opcao1`, `opcao2`, etc.) que contêm a lógica para realizar as operações relacionadas a cada opção do menu, passando os dados relevantes, como IDs de utilizador, dados dos restaurantes, entre outros, para executar as operações necessárias.

O loop while true garante que o menu continue a ser exibido até o utilizador escolher a opção de saída (`opcao6`), momento em que a aplicação termina.

Essencialmente, este código cria um ambiente interativo que permite ao utilizador explorar diferentes funcionalidades de um sistema de recomendação de restaurantes.

## 3.2 opcao1

A função `opcao1` é responsável por listar os restaurantes avaliados por um utilizador específico, identificado pelo `userID`.

```
function opcao1(userID, udata, rest)
    for i = 1:length(udata)
        if(udata(i,1) == userID)
            restaurantID = udata(i,2);
            restName = rest{restaurantID, 2};
            concelho = rest{restaurantID, 3};
            fprintf("ID: %-5d Nome: %-25s Concelho: %-20s\n",
   restaurantID, restName, concelho);
        end
    end
    disp('␣');
end
```

Ao percorrer a matriz `udata`, a função verifica as avaliações registadas para o utilizador cujo ID foi fornecido. Se encontrar avaliações desse utilizador, extrai o ID do restaurante avaliado a partir da segunda coluna da matriz `udata`.

Com o ID do restaurante em mãos, a função procura através dos dados dos restaurantes as informações correspondentes a esse restaurante, como nome e concelho.

## 3.3 opcao2

A função `opcao2` é responsável por encontrar o conjunto de restaurantes avaliados pelo utilizador mais similar.

```
function opcao2(currentUser, userData, numHashFuncs, unique_users,
   signatures, rest)
    % Find the index of the current user in the unique_users array
    currentUserIndex = find(unique_users == currentUser, 1);
```

```
% Validate if the currentUser is found
if isempty(currentUserIndex)
    fprintf('Current user ID not found.\n');
    return;
end

currentUserSignature = signatures(currentUserIndex, :);
maxSimilarity = 0;
mostSimilarUser = NaN;

for userIndex = 1:length(unique_users)
    if unique_users(userIndex) == currentUser
        continue;
    end

    userSignature = signatures(userIndex, :);
    similarity = sum(currentUserSignature == userSignature) /
numHashFuncs;

    if similarity > maxSimilarity
        maxSimilarity = similarity;
        mostSimilarUser = unique_users(userIndex);
    end
end
% Step 3: Find the most similar user
if isnan(mostSimilarUser)
    fprintf('No similar user found.\n\n');
else
    fprintf('Most similar user to %d is %d.\n', currentUser,
mostSimilarUser);
    similarUserRestaurants = userData(userData(:, 1) ==
mostSimilarUser, 2);
    disp('Restaurants evaluated by the most similar user:');

    for i = 1:length(similarUserRestaurants)
        restaurantID = similarUserRestaurants(i);

        rowIndex = find([rest{:, 1}] == restaurantID, 1);

        if ~isempty(rowIndex)
            disp(rest{rowIndex, 2});
        else
            disp(['Restaurant ID ' num2str(restaurantID) ' not
found.']);
        end
    end
```

```
    end
    disp('␣');
end
```

O processo começa por encontrar o índice do utilizador atual (`currentUser`) no array `unique_users`. Em seguida, a função calcula a assinatura do utilizador atual (`currentUserSignature`) e procura pelo utilizador mais similar ao atual, através da comparação entre assinaturas dos utilizadores.

A similaridade entre as assinaturas é calculada e o utilizador com a maior similaridade é identificado como o utilizador mais similar ao atual.

Se nenhum utilizador similar for encontrado, a função exibe uma mensagem a indicar que nenhum utilizador similar foi encontrado. Caso contrário, ela imprime o ID do utilizador mais similar e lista os restaurantes avaliados por esse utilizador.

## 3.4 opcao3

A função `opcao3` permite aos utilizadores pesquisar um prato especial em restaurantes com base em uma string inserida.

```
function opcao3(restaurants, minhashShingles, shingle_size, shingles_k)
    userInput = input('Write␣a␣string:␣', 's');

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

    MinHashString = inf(1, shingles_k);
    for j = 1:length(shinglesIn)
        chave = char(shinglesIn{j});
        hash = zeros(1, shingles_k);
        for x = 1:shingles_k
            chave = [chave num2str(x)];
            hash(x) = DJB31MA(chave, 127);
        end
        MinHashString(1,:) = min([MinHashString(1, :); hash]);
    end

    jaccardDistances = ones(1, size(restaurants, 1));

    for i=1:size(restaurants, 1)
        jaccardDistances(i) = sum(minhashShingles(i, :) ~=
   MinHashString) / shingles_k;
    end
```

```
    % Filter and sort the results
    [sortedDistances, sortedIndices] = sort(jaccardDistances);
    sortedRestaurants = restaurants(sortedIndices, :);

    % Display top 5 results with Jaccard distance <= 0.99
    disp('Top matching restaurants:');
    numResults = 0;
    for i = 1:length(sortedDistances)
        if sortedDistances(i) <= 0.99 && numResults < 5
            disp(['Name: ' sortedRestaurants{i, 2} ', Location: '
   sortedRestaurants{i, 3} ...
                ', Dish: ' sortedRestaurants{i, 6} ', Jaccard
   Distance: ' num2str(sortedDistances(i))]);
            numResults = numResults + 1;
        end
    end

    if numResults == 0
        fprintf('No matching restaurants found.\n');
    end
    disp(' ');
end
```

Inicialmente, o utilizador é solicitado a inserir uma string representando o prato especial que deseja procurar nos restaurantes.

A função então divide essa string em shingles de tamanho especificado (`shingle_size`) e calcula as representações de minhash para esses shingles. Isso é feito usando funções de hash para gerar valores de minhash que representam os shingles.

Em seguida, a função calcula as distâncias de Jaccard entre os shingles fornecidos pelo utilizador e os shingles pré-calculados, na data structure, dos restaurantes, com base nas representações de minhash. Isso permite encontrar a similaridade entre os shingles do utilizador e os shingles dos restaurantes.

Os resultados são filtrados e ordenados com base nas distâncias de Jaccard calculadas e no final a função exibe os cinco principais restaurantes cujos pratos especiais têm maior proximidade à string inserida pelo utilizador. Esses resultados são apresentados juntamente com o nome do restaurante, a localização, o prato especial e a distância de Jaccard associada.

Se nenhum restaurante corresponder ao nível mínimo de similaridade exigida, a função informa que nenhum restaurante correspondente foi encontrado.

## 3.5 opcao4

A função `opcao4` identifica os restaurantes mais similares a um restaurante específico avaliado por um determinado utilizador.

```
function opcao4(userID, udata, rest, minhashShingles, shingle_size,
    shingles_k, avgRatings)
     visitedIds = cell(1,length(udata(userID,2)));
     ind = 1;
     for i = 1:length(udata)
         if(udata(i,1) == userID)
             restaurantID = udata(i,2);
             restName = rest{restaurantID, 2};
             concelho = rest{restaurantID, 3};
             visitedIds{ind} = restaurantID;
             ind = ind + 1;
             fprintf("ID: %-5d Nome: %-25s Concelho: %-20s\n",
    restaurantID, restName, concelho);
         end
     end

     chosenRestaurantID = input('Insert the ID of the desired restaurant
    : ');

     if(ismember(chosenRestaurantID, [visitedIds{:}]))
             tipoCozinha = rest{chosenRestaurantID, 5};
             pratoTipico = rest{chosenRestaurantID, 6};
             if(ismissing(tipoCozinha))
                 tipoCozinha = '';
             end
             if(ismissing(pratoTipico))
                 pratoTipico = '';
             end
             evaluatedString = lower([tipoCozinha pratoTipico]);

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

         MinHashString = inf(1, shingles_k);
         for j = 1:length(shinglesIn)
             chave = char(shinglesIn{j});
             hash = zeros(1, shingles_k);
             for x = 1:shingles_k
                 chave = [chave num2str(x)];
                 hash(x) = DJB31MA(chave, 127);
             end
             MinHashString(1,:) = min([MinHashString(1, :); hash]);
         end
```

```
    jaccardDistances = ones(1, size(rest, 1));

    for i = 1:size(rest, 1)
        if(i == chosenRestaurantID)
            continue
        end
        jaccardDistances(i) = sum(minhashShingles(i, :) ~=
MinHashString) / shingles_k;
    end

    restaurants = mink(jaccardDistances, 3);
    highestDistance = max(restaurants);
    untieCount = sum(restaurants == highestDistance);
    restaurantsID = [];
    for i = 1:3
        restaurantsID = [restaurantsID find(jaccardDistances ==
restaurants(i))];
        if(length(restaurantsID) > 3)
            untieRestaurants = find(jaccardDistances ==
highestDistance);
            % Apply level 2
            toAdd = [];
            for j = 1:untieCount
                bestRestaurantID = resolveTiesWithRating(
untieRestaurants, avgRatings);
                toAdd = [toAdd bestRestaurantID];
                untieRestaurants(untieRestaurants ==
bestRestaurantID) = [];
            end
            if untieCount < 3
                restaurantsID = [restaurantsID(1,1:3-untieCount)
toAdd];
            else
                restaurantsID = toAdd;
            end
            break
        elseif(length(restaurantsID) == 3)
            break
        end
    end
    disp('Suggested␣Restaurants:␣')
    for i = 1:length(restaurantsID)
        restaurantID = restaurantsID(i);
        restName = rest{restaurantID, 2};
        concelho = rest{restaurantID, 3};
```

```
            tipoCozinha = rest{restaurantID, 5};
            pratoTipico = rest{restaurantID, 6};
            if(ismissing(pratoTipico))
                pratoTipico = 'Em falta';
            end
            fprintf("ID: %-5d Nome: %-30s Concelho: %-20s Tipo de
    Cozinha: %-25s Prato Tipico: %-10s\n", restaurantID, restName,
    concelho, tipoCozinha, pratoTipico);
        end


    else
        disp('The given ID does not belong to any of your visited
    restaurants');
    end
    disp(' ');
end

function bestRestaurantID = resolveTiesWithRating(restaurants,
    avgRatings)
    [~, sortedIndices] = sort(avgRatings(restaurants,2), 'descend');
    bestRestaurantID = avgRatings(restaurants(sortedIndices(1)),1);
end
```

Primeiramente, a função lista os restaurantes previamente visitados pelo utilizador com base no `userID` fornecido. O utilizador é solicitado a inserir o ID de um dos seus restaurantes visitados ao qual deseja encontrar outros restaurantes similares.

Se o restaurante escolhido estiver entre os visitados pelo utilizador, a função processa o tipo de cozinha e o prato típico desse restaurante para formar uma string de avaliação (`evaluatedString`). Essa string é dividida em shingles de tamanho específico (`shingle_size`), e são geradas representações de minhash para esses shingles.

A função então calcula a distância de Jaccard entre os shingles do restaurante escolhido e os shingles dos outros restaurantes. Com base nessa distância, identifica os três restaurantes mais similares ao restaurante escolhido e apresenta-os ao utilizador.

Se houver empate entre as distâncias de Jaccard dos restaurantes mais similares, a função utiliza as avaliações médias dos restaurantes (`avgRatings`) para desempatar, apresentando os restaurantes mais relevantes e similares ao escolhido pelo utilizador.

## 3.6 opcao5

A função `opcao5` em MATLAB é projetada para estimar o número de avaliações feitas por um utilizador, utilizando um filtro de Bloom.

```
function estimatedEvaluations = opcao5(user_data, bloomFilter)
    % Get the ID of the user whose evaluations are to be estimated
    userID = input('Enter the ID of the user: ');

    % Check if the entered ID is valid
    if ~any(user_data(:, 1) == userID)
        fprintf('Invalid user ID.\n\n');
        estimatedEvaluations = 0;
        return;
    end

    estimatedEvaluations = min(checkMembership(bloomFilter, userID));

    % Display the estimated number of evaluations
    fprintf('Estimated number of evaluations by user %d: %d\n\n',
   userID, estimatedEvaluations);
end
```

A seguir, apresenta-se uma explicação simplificada do código:

1. **Obter o ID do Utilizador:** A função inicia solicitando ao utilizador para inserir o ID do utilizador cujas avaliações deseja estimar.
2. **Verificar a Validade do ID:** O código verifica se o ID inserido existe nos dados do utilizador (`user_data`). Se o ID não for válido, a função exibe uma mensagem de erro e retorna zero.
3. **Estimar Avaliações:** Utiliza-se o filtro de Bloom (`bloomFilter`) para estimar o número de avaliações feitas pelo utilizador. O filtro de Bloom é uma estrutura de dados que permite fazer essa estimativa de forma eficiente.
4. **Exibir o Número Estimado de Avaliações:** Por fim, a função exibe o número estimado de avaliações feitas pelo utilizador com o ID fornecido.

Esta função é particularmente útil para estimar rapidamente o número de avaliações de um utilizador em um grande conjunto de dados, aproveitando a eficiência de um filtro de Bloom.

# Capítulo 4

# Notas Finais

Em todas as funções que necessitavam de uma função de dispersão (`MinHash` e `Bloom Filter`) utilizou-se a função já fornecida `DJB31MA.m`, usando o método de acrescentar progressivamente *strings* no final de uma outra string (alvo de *Hashing*) permite obter um determinado número de funções de dispersão diferentes.

De forma a aumentar a precisão do resultados obtidos quando o utilizador pretende procurar um prato especial, todas as letras foram convertidas para minúsculas.

As funções **MATLAB** desenvolvidas focam-se em tarefas específicas como estimar avaliações de utilizadores usando filtros de Bloom (`opcao5`), encontrar restaurantes similares com base nas preferências do utilizador (`opcao4`). Estas funções demonstram aplicações práticas no processamento de dados e implementação algorítmica, particularmente no contexto de sistemas de informação e motores de busca/recomendação.

Esta abordagem não apenas oferece funcionalidades específicas, mas também considera a experiência do utilizador ao alertar sobre possíveis situações de erro ou ausência de dados, melhorando a interação entre sistema e utilizador.