deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Relatório Trabalho (Restaurantes)

Professor: Carlos Bastos

Trabalho realizado por:

Bruno Tavares, nº113372

Diogo Costa, nº112714

Índice

Introdução

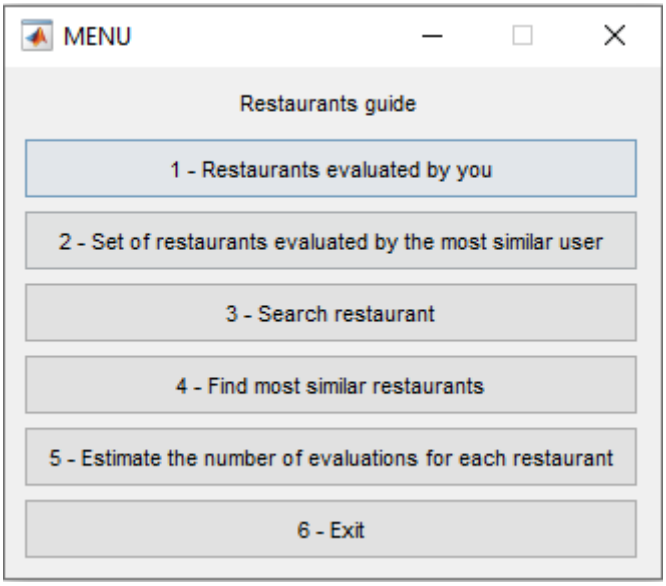
No contexto da disciplina de MPEI (Métodos Probabilísticos para Engenharia Informática), foi nos proposto a realização de um trabalho prático sobre algoritmos probabilísticos pelo professor Carlos Bastos. Temos como objetivo o desenvolvimento de uma aplicação em Matlab, com funcionalidades de um sistema online de disponibilização de informação e sugestão de restaurantes no norte de Portugal.

Resumidamente a aplicação começa por pedir o ID de um cliente para que depois consiga aplicar as funções desejadas, das quais iremos falar neste relatório pormenorizadamente.

Menu

Demos início à implementação da nossa aplicação começando por pedir ao utilizador o ID do cliente desejado. Para isso tivemos antes de saber quantos clientes havia no total, esta informação era necessária para que conseguíssemos verificar se o ID que o utilizador forneceu estaria certo ou errado, caso estivesse mal irá aparecer uma mensagem de erro.

Para a criação do menu usamos uma função que o MATLAB disponibiliza, **menu().**



Carregar informação de ficheiros

```
turistas1 = load("turistas1.data");
turistas = turistas1(:, [1, 2, 4]); % descarta a 3ª coluna

users = unique(turistas(:,1));
numUsers= length(users); % número de clientes

rest = readcell('restaurantes.txt', 'Delimiter', '\t');
numRest = height(rest); % número de restaurantes
```

Primeiramente executamos load ao “turistas1.data” que na primeira coluna identifica o turista, na segunda os restaurantes que cada utilizador avaliou e por fim na coluna 4 contém o score atribuído ao restaurante (**ignoramos** a coluna 3 pois tinha informação que não influenciava em nenhuns dos “cases”)

Também selecionamos os users “uniques” para termos a informação da **quantidade de turistas existentes**. Por fim lemos o ficheiro “restaurantes.txt” com delimitação por “tabs”, e retiramos o número de linhas desse ficheiro que corresponde ao **número total de restaurantes** (numRest).

Opção 1 (Restaurants evaluated by you)

O Case 1 do código está encarregue de exibir os restaurantes avaliados por um usuário específico.

```
case 1
    % Get evaluations from the specific user
    user_restaurants = sort(restaurants{user_id});
    % Check if the user has evaluations
    if isempty(user_restaurants)
        fprintf('This user has not evaluated any restaurant.\n');
    else
        fprintf('Evaluated restaurants by user ID=%d\n', user_id)
        fprintf('ID\t|\tRestaurant Name\t|\tDistrict\n');
        for i = 1:length(user_restaurants)
            restID = user_restaurants(i);
            restInfo = rest(restID, [1, 2, 4]);
            fprintf('%s\t|\t%s\t|\t%s\n', num2str(restID), char(restInfo{2}), char(restInfo{3}));
        end
    end
    clear user_restaurants;
    clear restInfo;
    clear restID;
    fprintf('\n');
```

A variável **user_restaurants** contém os IDs dos restaurantes avaliados pelo utilizador, ordenados, caso o user não tenha avaliado nenhum restaurante a função isempty verifica isso e transmite uma mensagem de erro.

Se o user avaliou um ou mais restaurantes, as informações dos restaurantes são exibidas desta forma (ID, Nome do Restaurante, Concelho), para isso temos um loop “for” que percorre a lista de restaurantes avaliados pelo usuário e depois usando **rest,** conseguimos obter as informações específicas sobre cada restaurante.

Por fim as variáveis temporárias são limpas para evitar conflitos em execuções subsequentes.

Opção 2 (Set of restaurants evaluated by the most similar user)

O Case 2 implementa um algoritmo simples que lista o conjunto de restaurantes avaliados pelo utilizador mais "similar" ao utilizador atual.

```
case 2
    max_similarity = 0;
    most_similar_user_id = 0;

    for other_user_id = 1:numUsers
        if other_user_id == user_id
            continue;
        end

        user_restaurants = sort(restaurants{user_id});
        other_user_restaurants = sort(restaurants{other_user_id});

        intersection = intersect(user_restaurants, other_user_restaurants);
        union_set = union(user_restaurants, other_user_restaurants);
        similarity = length(intersection) / length(union_set);

        if similarity > max_similarity
            max_similarity = similarity;
            most_similar_user_id = other_user_id;
        end
    end

    if most_similar_user_id > 0
        most_similar_user_restaurants = sort(restaurants{most_similar_user_id});
        fprintf("Evaluated retaurants by the most similar user to ID=%d\n", user_id)
        fprintf('ID\t|\tRestaurant Name\t|\tDistrict\n');
        for i = 1:length(most_similar_user_restaurants)
            restID = most_similar_user_restaurants(i);
            restInfo = rest(restID, [1, 2, 4]);
            fprintf('%s\t|\t%s\t|\t%s\n', num2str(restID), char(restInfo{2}), char(restInfo{3}));
        end
    else
        fprintf('No more similar user found.\n');
    end
```

O (**max_similarity**) armazena a similaridade máxima encontrada durante a iteração e o (**most_similar_user_id**) guarda o ID do usuário mais similar.

O loop "for" itera sobre todos os usuários (**other_user_id**) exceto para o user atual (**user_id**). Os restaurantes avaliados pelo user_id e pelo other_user_id são obtidos e ordenados. A interseção e a união desses conjuntos também são calculadas.

A similaridade **Jaccard** é calculada pelo tamanho da interseção dividido pelo tamanho da união, se a similaridade calculada for maior que a similaridade máxima encontrada até agora (**max_similarity**), a similaridade máxima e o ID do user mais similar são atualizados.

Após o término do loop, verifica-se se um usuário mais similar foi encontrado (**most_similar_user_id > 0**), caso seja encontrado, os restaurantes avaliados desse user são impressas na tela (ID, nome do restaurante e concelho).

Opção 3 (Search restaurant)

O Case 3 do código é responsável por realizar uma pesquisa por restaurantes com base em uma string fornecida pelo user.

```
case 3
    parameter = lower(input('Write a string: ', 's'));
    while (length(parameter) < shingleSize)
        fprintf('Write a minimum of %d characters\n', shingleSize);
        parameter = lower(input('Write a string: ', 's'));
    end
    searchRest(parameter, restMinHash, numHash, restNames, shingleSize)
    fprintf('\n');
    clear parameter;
    fprintf('\n');
```

Começamos por obter a string de pesquisa do user utilizando a função input para a receber, de seguida convertemo-la para minúsculas usando lower e guardamos em "**parameter**".

Depois temos um loop para garantir que a string de pesquisa tenha pelo menos o tamanho especificado por **shingleSize** (colocamos o valor 3). De seguida chamamos a função **searchRest** para realizar a pesquisa com base na string fornecida pelo usuário, bem como nas estruturas de dados **restMinHash** e **restNames**.

A função **searchRest** é responsável por calcular o **MinHash** para a string de pesquisa e encontrar os restaurantes mais similares com base nesse **MinHash**.

Função searchRest()

```
function searchRest(parameter, restMinHash, numHash, restaurants, shingleSize)
    minHashSearch = inf(1, numHash);
    for j = 1 : (length(parameter) - shingleSize + 1)
        shingle = char(parameter(j:(j+shingleSize-1)));
        h = zeros(1, numHash);
        for i = 1 : numHash
            shingle = [shingle num2str(i)];
            h(i) = DJB31MA(shingle, 127);
        end
        minHashSearch(1, :) = min([minHashSearch(1, :); h]);
    end

    threshold = 0.99;
    [similarTitles,distancesRest,k] = filterSimilar(threshold,restaurants,restMinHash,minHashSearch,numHash);

    if (k == 0)
        disp('No results found');
    elseif (k > 5)
        k = 5;
        fprintf('Most similar titles to "%s":\n', parameter);
    end

    distances = cell2mat(distancesRest);
    [distances, index] = sort(distances);

    for h = 1 : k
        fprintf('%s - Distância: %.3f\n', similarTitles{index(h)}, distances(h));
    end
end
```

Esta função começa por iterar sobre os shingles da string de pesquisa e calcula o MinHash correspondente, depois utilizando a função **filterSimilar** encontra os restaurantes com MinHash semelhante à string de pesquisa.

Por fim, se não houver resultados, exibe uma mensagem informando que nenhum resultado foi encontrado, se houver mais de 5 resultados, limita a exibição a 5 resultados e mostra os restaurantes mais similares à string e as distâncias correspondentes.

Função filterSimilar()

```
function [similarRest,restDistances,k] = filterSimilar(threshold,restaurants,restMinHash,minHash_search,numHash)
    similarRest = {};
    restDistances = {};
    numTitles = length(restaurants);
    k=0;
    for n = 1 : numTitles
        distancia = 1 - (sum(minHash_search(1, :) == restMinHash(n,:)) / numHash);
        if (distancia < threshold)
            k = k+1;
            similarRest{k} = restaurants{n};
            restDistances{k} = distancia;
        end
    end
end
```

A **filterSimilar()** começa por inicializar a **similarRest** e a **restDistances** como células vazias, **numTitles** recebe o número total de restaurantes no conjunto de dados.

De seguida dentro do "for", a distância de **Jaccard** é calculada comparando os MinHashes da string fornecidida (**minHash_search**) com os MinHashes do restaurante atual (**restMinHash(n, :)**). A distância calculada é comparada com o limiar especificado (**threshold**). Se a distância for inferior ao limiar, o restaurante é considerado similar.

A função retorna as células contendo os **restaurantes similares** (similarRest), as **distâncias** correspondentes (restDistances) e o **número total de restaurantes similares** (k).

Opção 4 (Find some similar restaurants)

O Case 4 lida com a escolha de um restaurante específico por parte do user e apresenta os três restaurantes mais similares ao escolhido.

```
case 4
    user_restaurants = sort(restaurants{user_id});
    if isempty(user_restaurants)
        fprintf('This user has not evaluated any restaurant.\n');
    else
        fprintf('Evaluated restaurants by user ID=%d\n', user_id)
        fprintf('ID\t|\tRestaurant Name\t|\tDistrict\n');
        for i = 1:length(user_restaurants)
            restID = user_restaurants(i);
            restInfo = rest(restID, [1, 2, 4]);
            fprintf('%s\t|\t%s\t|\t%s\n', num2str(restID), char(restInfo{2}), char(restInfo{3}));
        end
        fprintf('\n');
        verifier = 0;
        while verifier == 0
            chosenRestID = input("Enter the ID of the chosen restaurant: ", "s");
            chosenRestID = str2double(chosenRestID);
            for i = 1:length(user_restaurants)
                restID = user_restaurants(i);
                if chosenRestID == restID
                    verifier = 1;
                    restInfo = rest(restID, [1, 2, 4]);  % Update restInfo for the chosen restaurant
                    break;
                end
            end
            if verifier == 0
                fprintf("ERROR: Enter a valid restaurant ID\n");
            end
        end
    end
fprintf('Chosen restaurant:\n%s\t|\t%s\t|\t%s\n', num2str(restID), char(restInfo{2}), char(restInfo{3}));
getTopSimilarRestaurants(rest, restInfo, restNames, restMinHash, numHash, shingleSize);
```

A opção 4 começa por obter avaliações dos users na variável **user_restaurants** que é uma lista ordenada dos restaurantes avaliados pelo user_id, se o utilizador não tiver avaliações, uma mensagem informativa é exibida.

Os restaurantes avaliados pelo user são apresentados na tela, incluindo os campos de ID, nome do restaurante e concelho, de seguida o **user é solicitado a inserir o ID do restaurante** que deseja explorar, é feita uma verificação para garantir que o ID do restaurante inserido seja **válido** e correspondente a um restaurante avaliado por esse user.

Por fim a função **getTopSimilarRestaurants()** é chamada para encontrar os três restaurantes mais similares ao restaurante escolhido.

Função getTopSimilarRestaurants()

```
function getTopSimilarRestaurants(rest, chosenRestInfo, restNames, restMinHash, numHash, ~)
    restaurantName = char(chosenRestInfo{2});
    nameIndex = find(strcmp(restNames, restaurantName));

    if ~isempty(nameIndex)
        selectedRestMinHash = restMinHash(nameIndex, :);

        [similarRest, distances, ~] = filterSimilar(0.99, restNames, restMinHash, selectedRestMinHash, numHash);

        chosenIndex = find(strcmp(similarRest, restaurantName));
        if ~isempty(chosenIndex)
            similarRest(chosenIndex) = [];
            distances(chosenIndex) = [];
        end

        [~, idx] = sortrows([distances']);

        fprintf('\nThe 3 most similar restaurants:\n');
        fprintf('ID\t|\tRestaurant Name\t|\tDistrict\n');
        for k = 1:min(3, length(similarRest))
            similarRestInfo = rest(strcmp(rest(:, 2), similarRest{idx(k)}), [1, 2, 4]);
            fprintf('%s\t|\t%s\t|\t%s\n', num2str(similarRestInfo{1}), char(similarRestInfo{2}), char(similarRestInfo{3}) );
        end
    else
        error('Restaurant name not found.');
    end
end
```

A função começa por obter o índice do restaurante escolhido na lista de nomes de restaurantes (**restNames**). Após isso, verifica se o índice não está vazio para garantir que o restaurante escolhido esteja presente na lista de restaurantes.

Depois, a função recolhe as informações de MinHash específicas do restaurante escolhido a partir da matriz global de MinHash, que guarda essas informações para todos os restaurantes.

Utilizando a função **filterSimilar** (já explicada na case 3), são identificados os restaurantes mais similares com base na distância de MinHash. O vetor MinHash do restaurante escolhido é comparado com os restantes para calcular as distâncias e selecionar os restaurantes que atendem a um determinado limiar de similaridade.

Para evitar a apresentação do próprio restaurante escolhido na lista de resultados, o mesmo é removido da lista. Posteriormente, os resultados são ordenados com base na distância para apresentar os restaurantes mais similares primeiro.

Por fim, a função imprime na tela os **três** restaurantes **mais similares**, incluindo informações como ID, nome do restaurante e concelho. No caso em que o nome do restaurante escolhido não tenha nenhum similar, a função emite um erro indicando que o nome do restaurante não foi encontrado.

Opção 5 (Estimate the number of evaluations for each restaurant)

```
case 5
    restaurantID = 0;
    while(restaurantID <= 0 || restaurantID > 213)
        restaurantID = input('Enter Restaurant ID: ');
        if (restaurantID < 1 || restaurantID > 213)
            fprintf("ERROR: Enter a valid Restaurant ID\n");
        end
    end

    restaurantIDStr = num2str(restaurantID);
    totalAvaliacoes = 0;
    for userID = 1:numUsers
        user_restaurants = sort(restaurants{userID});
        if ismember(restaurantID, user_restaurants)
            filtro = adicionarElemento(filtro, restaurantIDStr, numHashFunc, filterSize);
            totalAvaliacoes = totalAvaliacoes + 1;
        end
    end

    fprintf("Total number of reviews for Restaurant ID %d: %d\n", restaurantID, totalAvaliacoes);
```

O código começa pedindo ao user que insira o ID do restaurante que tem de ser maior que 0 e menor que 213 (número total de restaurantes). Inicializa-se uma string (**restaurantIDStr**) com o ID do restaurante convertido para string e também se cria uma variável **totalAvaliacoes** para 0, que será usada para contar o número total de avaliações para o restaurante.

Itera-se sobre todos os users (**userID**) para verificar se o restaurante escolhido está presente nas suas avaliações. Se o restaurante estiver presente na lista de avaliações do utilizador, a função **adicionarElemento** é chamada para atualizar um filtro específico relacionado a esse restaurante. Depois do loop "for" imprime-se na tela o total de avaliações encontradas para o restaurante específico.

Funções auxiliares

- inicializarFiltro()

```
function filtro = inicializarFiltro(n)
    filtro = zeros(n, 1);
end
```

- adicionarElemento()

```
function filtro = adicionarElemento(filtro, chave, numHashFunc, tablesize)
    for i = 1:numHashFunc
        chave1 = [chave num2str(i)];
        code = mod(DJB31MA(chave1, 127), tablesize) + 1;
        filtro(code) = filtro(code) + 1;
    end
end
```

- pertenceConjunto()

```
function resultado = pertenceConjunto(filtro, chave, numHashFunc, tablesize)
    res = zeros(numHashFunc, 1);
    for i = 1:numHashFunc
        chave1 = [chave num2str(i)];
        code = mod(DJB31MA(chave1, 127), tablesize) + 1;
        res(i) = filtro(code);
    end
    resultado = sum(res);
end
```

Visão geral do trabalho

O nosso trabalho é constituído por dois ficheiros (main.m e functions.m).

Main.m

O ficheiro **main.m** é o ponto central da nossa aplicação, contendo a lógica principal para interação com o user e a execução das principais funcionalidades do programa, é neste ficheiro que está presente o **switch-case**, que é responsável por direcionar o fluxo do programa com base na escolha do utilizador no menu. Cada caso contém o código associado à operação específica relacionada à escolha do user e também dentro de cada opção, são feitas **chamadas de funções** ou operações específicas para executar a funcionalidade desejada.

Functions.m

Já o ficheiro **functions.m** é um arquivo auxiliar que contem funções relacionadas à manipulação de dados e à implementação de algoritmos chave do sistema, como por exemplo os MinHash:

- **minHashUsers(users, numHash, restaurants)**: Para cada restaurante avaliado pelo user, gera hashes utilizando a função de dispersão DJB31MA e seleciona o

mínimo. Retorna uma matriz em que cada linha representa um user e as colunas representam os valores MinHash associados.

- **getDistancesMinHashUsers(numUsers, matrizMinHash, numHash):** Calcula as distâncias entre os users com base nas suas MinHash.
- **minHashRest(restList, numHash, shingleSize):** Calcula a MinHash para cada restaurante com base nos seus shingles
- **getDistancesRest(numRest, MinHashRest, numHash):** Calcula as distâncias entre os restaurantes com base na MinHash.
- **DJB31MA:** Esta função é usada para criar valores hash únicos e distribuídos para diferentes sequências de caracteres (chave).

Resultados para cada caso (ID 1)

Case 1:

```
>> main
Insert User ID (1 to 836): 1
Evaluated restaurants by user ID=1
ID  |   Restaurant Name |   District
18  |   Outerinho   |   Barcelos
21  |   Casa de Carvalhelhos (Taberna)  |   Boticas
22  |   Casa de Vilar (Taberna) |   Boticas
40  |   O Rochedo   |   Bragança
44  |   Churraqueira o Paço |   Cabeceiras de Basto
123 |   A Cozinha da Terra  |   Paredes
134 |   Encanada    |   Ponte de Lima
172 |   Casa do Ermeiro (Taberna)   |   Valpaços
198 |   Lameirão    |   Vila Real
205 |   Churrasqueira Barros    |   Vila Verde
206 |   Manjar do Mar   |   Vila Verde
```

Case 2:

```
Evaluated retaurants by the most similar user to ID=1
ID  |   Restaurant Name |   District
12  |   Grill/Costa do Vez  |   Arcos de Valdevez
21  |   Casa de Carvalhelhos (Taberna)  |   Boticas
43  |   Solar Bragançano    |   Bragança
89  |   Picadeiro   |   Guimarães
134 |   Encanada    |   Ponte de Lima
134 |   Encanada    |   Ponte de Lima
165 |   Portobello  |   V. Praia Âncora
172 |   Casa do Ermeiro (Taberna)   |   Valpaços
198 |   Lameirão    |   Vila Real
```

Case 3 (videira):

```
Write a string: videira
Most similar titles to "videira":
Videira - Distance: 0.000
Bagoeira - Distance: 0.730
A Lareira - Distance: 0.830
Salgueira - Distance: 0.840
Picadeiro - Distance: 0.850
```

Case 4:

```
Evaluated restaurants by user ID=1
ID  |   Restaurant Name |   District
18  |   Outerinho   |   Barcelos
21  |   Casa de Carvalhelhos (Taberna)  |   Boticas
22  |   Casa de Vilar (Taberna) |   Boticas
40  |   O Rochedo   |   Bragança
44  |   Churraqueira o Paço |   Cabeceiras de Basto
123 |   A Cozinha da Terra  |   Paredes
134 |   Encanada    |   Ponte de Lima
172 |   Casa do Ermeiro (Taberna)   |   Valpaços
198 |   Lameirão    |   Vila Real
205 |   Churrasqueira Barros    |   Vila Verde
206 |   Manjar do Mar   |   Vila Verde

Enter the ID of the chosen restaurant: 18
Chosen restaurant:
18  |   Outerinho   |   Barcelos

The 3 most similar restaurants:
ID  |   Restaurant Name |   District
78  |   Casa Outeirinho |   Famalicão
45  |   Luis do Outeirinho  |   Cabeceiras de Basto
149 |   Pelourinho  |   Régua
```

Case 5:

```
Enter Restaurant ID: 1
Total number of reviews for Restaurant ID 1: 68
```

Resultados para case 1 e 2 (ID: 210)

Case 1:

```
>> main
Insert User ID (1 to 836): 210
Evaluated restaurants by user ID=210
ID  |   Restaurant Name |   District
8   |   Pousada Santa Maria do Bouro    |   Amares
9   |   Quinta do Esquilo   |   Amares
25  |   Arcoense    |   Braga
43  |   Solar Bragançano    |   Bragança
45  |   Luis do Outeirinho  |   Cabeceiras de Basto
47  |   A Gaivota   |   Caminha
47  |   A Gaivota   |   Caminha
54  |   Churrascaria Veiga  |   Carrazeda de Ansiães
100 |   Pensão Magalhães    |   Marco Canaveses
117 |   O Transmontano  |   Mondim de Basto
120 |   Nevada  |   Montalegre
123 |   A Cozinha da Terra  |   Paredes
144 |   Quebra-Mar  |   Póvoa de Varzim
169 |   Pousada S. Teotonico    |   Valença
191 |   Costa do Sol    |   Vila Pouca Aguiar
201 |   O Barriguinha Cheia |   Vila Real
```

Case 2:

```
Evaluated retaurants by the most similar user to ID=210
ID  |   Restaurant Name |   District
8   |   Pousada Santa Maria do Bouro    |   Amares
9   |   Quinta do Esquilo   |   Amares
21  |   Casa de Carvalhelhos (Taberna)  |   Boticas
31  |   Maia    |   Braga
52  |   Solar do Pescado    |   Caminha
54  |   Churrascaria Veiga  |   Carrazeda de Ansiães
59  |   Albergaria Borges ( Rest.Típico )   |   Chaves
81  |   O Ferrugem  |   Famalicão
81  |   O Ferrugem  |   Famalicão
81  |   O Ferrugem  |   Famalicão
113 |   A Lareira   |   Mogadouro
118 |   Tasca da Tia Alice  |   Mondim de Basto
119 |   Casa de Padornelos (Taberna)    |   Montalegre
120 |   Nevada  |   Montalegre
123 |   A Cozinha da Terra  |   Paredes
141 |   O Marinheiro    |   Póvoa de Varzim
145 |   Castas e Pratos |   Régua
150 |   Quinta de São Domingos  |   Régua
179 |   Hotel Forte S. João Baptista    |   Vila do Conde
191 |   Costa do Sol    |   Vila Pouca Aguiar
204 |   Terra da Montanha   |   Vila Real
206 |   Manjar do Mar   |   Vila Verde
```

Justificação para escolha de valores

Escolhemos o valor 100 para NumHash porque depois de experimentarmos com outros valores, acabamos por nos aperceber que o valor 100 é bom encontrar um equilíbrio entre a precisão da representação e a eficiência computacional, quando aumentamos este valor sentimos uma grande demora a correr o código, ainda assim, 100 funções de dispersão fornecem uma boa precisão para as distâncias de Jaccard.

No guião aconselharam a usar o tamanho do shingle entre 2 e 5 caracteres e acabamos por escolher 3 caracteres. Quanto maior for o tamanho do shingle, menor vai ser a probabilidade de haver correspondência com outros shingles, mas também aumenta a dimensionalidade do espaço de hash e a complexidade computacional. Por estas razões achamos bem escolher um valor médio, não muito baixo e nem alto demais, depois de vários testes entre os valores 3 e 4, acabamos por optar pelo valor 3 para uma boa funcionalidade do programa.

Conclusão

Com a realização deste projeto no âmbito da disciplina de MPEI conseguimos melhorar os nossos conhecimentos quanto aos assuntos abordados no guião 4 (funções de dispersão, filtros Bloom…).

Acredito que também tenhamos melhorado o nosso conhecimento computacional com este trabalho, pois provavelmente agora estamos mais confortáveis em programar no Matlab.

Penso que os principais tópicos do trabalho tenham sido efetuados com sucesso e com clareza no código.