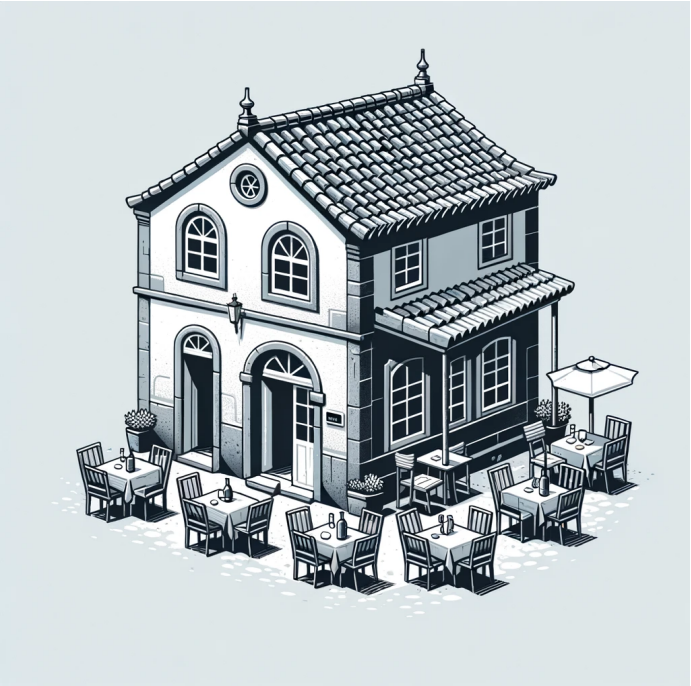
deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Relatório PL04 - Turma P2



Métodos Probabilísticos para Engenharia Informática

Prof. Carlos Alberto da Costa Bastos

Maria Luis Delgado Linhares	113534
Gabriel Martins Silva	113786

Aveiro, Dezembro 2023

Introdução

No âmbito da cadeira de MPEI (Métodos Probabilísticos para Engenharia Informática) foi-nos apresentada esta prática laboral acerca de algoritmos probabilísticos. Este trabalho tem como objetivo, desenvolver em Matlab, uma aplicação com algumas funcionalidades de um sistema online de disponibilização de informação e sugestão de restaurantes no norte de Portugal.

Este trabalho prático convida à aplicação de conceitos probabilísticos avançados no desenvolvimento de um sistema de recomendação que não apenas atenda às expectativas dos usuários por sugestões relevantes, mas que também processe as informações de maneira eficiente. A aplicação projetada utiliza um conjunto de dados que reflete as interações dos turistas com diferentes estabelecimentos, servindo como base para a implementação de algoritmos que incluem MinHash para avaliação de semelhança entre usuários e estabelecimentos e filtros de Bloom para contagem eficiente de avaliações de restaurantes.

O objetivo deste relatório é documentar todas as etapas de desenvolvimento da aplicação, desde a análise dos dados até a metodologia usada.

Inicialização

Inicialmente, inicializamos todas as estruturas de dados provenientes de correr o primeiro script (*dados.m*).

```
% load de ficheiros
load("turistas1.data")
rest = readcell('restaurantes.txt','Delimiter','\t');
load("RestaurantsByUser.mat");
load("assinaturasUsers.mat");
load("distanciasUsers.mat");
load("bloomFilter.mat");
```

Opções Implementadas

Opção 1

Nesta opção é pedido para listar todos os restaurantes que o utilizador atual (identificado pelo ID pedido inicialmente) visitou.

```
case 1
            restaurantes = Set{user_ID};
            listarRestaurantes(restaurantes,rest)
```

Set é proveniente do ficheiro `"RestaurantsByUser.mat"`.
`restaurantes ->` vai ficar com o ID dos restaurantes avaliados para cada turista
A função `listarRestaurantes()` serve para dar print de todos os restaurantes do utilizador atual (`user_ID`) com o seu ID (do restaurante), nome e concelho onde está localizado.

```
function listarRestaurantes(listaIDsRestaurantes,rest)
   for r = 1:length(listaIDsRestaurantes)
       restaurante = rest(listaIDsRestaurantes(r),:);
       fprintf("%-4d %-40s %-20s\n", restaurante{1}, restaurante{2},
restaurante{4});
   end
end
```

Opção 2

Nesta opção é listado o conjunto de restaurantes avaliados mais “similar” ao utilizador atual.

```
case 2
            distancias = dists(user_ID,:);
            minDist = 1;
            for u=1:Nu
                if distancias(u) < minDist && u ~= user_ID
                    minDist = distancias(u);
                    ind = u;
                end
            end
            restaurantes = Set{ind};
            listarRestaurantes(restaurantes,rest)
```

`dists` é proveniente do ficheiro `"distanciasUsers.mat"`.
`distancias ->` é um vetor com o tamanho do número de users, onde está contida a distância entre o user atual (`user_ID`) e todos os outros users

No for loop, percorremos o array e encontramos o user ao qual corresponde a menor distância, logo maior similaridade

`restaurantes = Set{ind} ->` vai buscar todos os restaurante avaliados pelo utilizador mais “similar” e a função `listarRestaurantes()` vai dar print dos mesmo

De modo a calcular as distâncias criamos uma função chamada de `MinHashCalculation()`, que realiza o cálculo das assinaturas MinHash para os conjuntos de turistas representados em Set através da função `calcularMatrizAssinaturas()`
e, em seguida, calcula as distâncias entre esses turistas com base nas assinaturas usando a função `calcularDistancias()` que calcula a distância de similaridade entre pares únicos de conjuntos usando assinaturas MinHash e armazena as distâncias em uma matriz chamada `distancias`. Isto vai permitir comparar o quão semelhantes são os conjuntos entre si.

Opção 3

Nesta opção são listados os 5 restaurantes com nome mais similares a uma string introduzida pelo utilizador.

```
case 3
            string = input("Write a string: ","s");

            sortedDists = StringsSimilares(rest(:,2),string,0.99);
            if length(sortedDists) > 5
                sortedDists = sortedDists(1:5,:);
            end

            if isempty(sortedDists)
                disp("Restaurant not found");
            end

            for r = 1:length(sortedDists(:,1))
                nome = rest{sortedDists(r,1),2};
                dist = sortedDists(r,2);
                fprintf("%-35s %.3f\n",nome,dist);
            end
```

Este código permite ao utilizador encontrar restaurantes semelhantes com base em uma string inserida e exibir na tela os 5 mais semelhantes. Para tal criamos a função `StringsSimilares()` que calcula a similaridade entre uma string de entrada e uma lista de strings (nomes) usando a técnica de MinHash.
Esta função extrai shingles de tamanho 2 dos nomes dos restaurantes e coloca-os num conjunto `shingles`. Escolhemos como o tamanho do shingle 2 e de número de funções de dispersão como 200 de modo a ter resultados mais precisos ter por base as conclusões tiradas aquando da resolução da secção 4.3 deste guião.

```
for s=1:length(nomes)
        nome = nomes{s};
        while length(nome) >= tamanhoShingle
            Shingle = nome(1:tamanhoShingle);
            nome = nome(2:length(nome));
            shingles{count} = Shingle;
            count = count + 1;
        end
end
shingles = unique(shingles);
```

De seguida, cria uma matriz de entrada que indica quais shingles estão presentes em cada string da lista de nomes.

```
matrizEntrada = zeros(length(shingles),length(nomes));
   for n = 1:length(nomes)
       for s = 1:length(shingles)
           if contains(nomes(n),shingles(s))
               matrizEntrada(s,n) = 1;
           end
       end
   end
```

Depois, calcula a similaridade entre a string de entrada e cada string da lista de nomes com base nas assinaturas MinHash usando a função `calcularMatrizAssinaturas()` já explicada anteriormente. Mantém as strings que têm uma distância menor ou igual ao do limite (threshold) especificado.

```
if (1-s) <= threshold
     SimilarNames(count,:) = [n 1-s];
     count = count + 1;
 end
```

Por fim, retorna as strings semelhantes ordenadas pela medida de similaridade em `sortedDists`.

```
SimilarNames = SimilarNames(1:end-1,:);
   sortedDists = [];
   if ~isempty(SimilarNames)
       sortedDists = sortrows(SimilarNames,2);
   end
```

Opção 4

A partir do restaurantes avaliados pelo utilizador atual, este vai escolher um deles e o programa vai listar os três mais similares a esse com base em vários parâmetros (localidade, concelho, tipo de cozinha, prato(s) recomendado(s) e dia(s) encerrado).

O programa começa por listar os restaurantes já avaliados pelo utilizador atual. De seguida, pedimos para escolher um deles onde verificamos se o restaurante escolhido está na lista de restaurantes do usuário.

```
listarRestaurantes(Set{user_ID},rest)

id = input("Choose a restaurant ID: ");
while ~ismember(id, Set{user_ID})
   disp("Restaurant not found for this user")
   id = input("Choose a restaurant ID: ");
end
```

Calcula a similaridade entre o restaurante escolhido e outros restaurantes em relação a vários campos, como localização, concelho,tipos de cozinha, pratos recomendados e dias

de encerramento. É calculada a distância para cada campo e de seguida a média de todas as distâncias de cada restaurante e de seguida são ordenados com base na média das distâncias por ordem crescente, pois quanto menor a distância maior a similaridade.
Caso a média não seja suficiente para classificar os restaurantes, há uma segunda fase, para os casos de empate, onde é feita a comparação utilizando o valor médio de avaliação dos restaurantes.

Antes de iniciar a segunda fase, comparamos os três primeiros valores entre si e o terceiro com o quarto, de modo a ver se há distâncias iguais e verificar a necessidade de entrar na segunda fase, atraves do tamanho do array `DistFinal`, criado para armazenar os valores finais onde o valores da 2ª coluna irão ser todos diferentes.

```
DistFinal = [];

    if d1 ~= d2
        DistFinal(1,1) = DistMedia(2,1);
        DistFinal(1,2) = d1;
        if d2 ~= d3
            DistFinal(2,1) = DistMedia(3,1);
            DistFinal(2,2) = d2;
            if d3 ~= DistMedia(5,2)
                DistFinal(3,1) = DistMedia(4,1);
                DistFinal(3,2) = d3;
            end
        end
    end
```

Dependendo do tamanho do array `DistFinal` vamos ter várias situações de igualdade.

- 1º caso: os primeiro dois valores são diferentes, mas o terceiro é igual a pelo menos a um valor. Todos os valores que são iguais são adicionados ao array `DistsIguais`.

```
    elseif length(DistFinal(:,1)) == 2
        d = d3;
        i = 4;
        while d == d3
            i = i + 1;
            d = DistMedia(i-1,2);
            if d ~= d3
                break
            end
            DistsIguais(i-4,1) = DistMedia(i-1,1);
            DistsIguais(i-4,2) = d;
        end
        mediasAval=calcularAvaliacaoMedia(turistas1,DistsIguais)
        DistFinal(3,:) = mediasAval(1,:);
```

- 2º caso: a distância dos primeiros dois valores é diferente, mas o segundo valor é igual ao terceiro, podendo haver mais valores igual a este. Usamos a função `calcularAvaliacaoMedia` de modo a ver qual do empates vai passar a ter prioridade.

```
elseif length(DistFinal(:,1)) == 1
      d = d2;
      i = 3;
      while d == d2
            i = i + 1;
            d = DistMedia(i-1,2);
            if d ~= d2
                  break
            end
            DistsIguais(i-3,1) = DistMedia(i-1,1);
            DistsIguais(i-3,2) = d;
      end
      mediasAval = calcularAvaliacaoMedia(turistas1,DistsIguais);
      DistFinal(2:3,:) = mediasAval(1:2,:);
end
```

- 3º caso: Quando o primeiro e o segundo valores são iguais, mas o segundo é diferente do terceiro. De modo a lidar com isto, inicialmente é calculada a avaliação média entre os dois primeiros e de seguido, são colocado no `DistFinal`. Depois, verificamos quantos valores iguais ao terceiro há.

```
            while d == d1
                  i = i + 1;
                  d = DistMedia(i-1,2);
                  if d ~= d1
                        break
                  end
                  DistsIguais(i-2,1) = DistMedia(i-1,1);
                  DistsIguais(i-2,2) = d;
            end
            mediasAval = calcularAvaliacaoMedia(turistas1,DistsIguais);
            DistFinal(3,:) = mediasAval(1,:);

            if d3 ~= DistMedia(5,2)
                  DistFinal(3,1) = DistMedia(4,1);
                  DistFinal(3,2) = d3;
            else
                  d = d3;
                  i = 4;
                  DistsIguais = [];
                  while d == d3
                        i = i + 1;
```

```
                    d = DistMedia(i-1,2);
                    if d ~= d3
                          break
                    end
                    DistsIguais(i-4,1) =
                    DistMedia(i-1,1);DistsIguais(i-4,2) = d;
              end
              mediasAval = calcularAvaliacaoMedia(turistas1,
        DistsIguais);
              DistFinal(3,:) = mediasAval(1,:);
```

Caso o terceiro e quarto valores sejam diferentes, o terceiro é adicionado ao `DistFinal`. Caso sejam iguais, voltamos a inicializar o array `DistsIguais`, e comparamos até encontrar um valor diferente. Depois, calculamos o valor médio da avaliação e comparamos, sendo que o que tiver maior avaliação tem prioridade e é adicionado ao array `DistFinal`.

- 4º caso: Pelo menos, os primeiros três valores serem iguais, por isso vamos comprar até encontrar um valor diferente. Após isto, calculamos a avaliação média, ordenamos por ordem decrescente e adicionamos ao array `DistFinal` os três primeiros elementos.

```
d = d1;
      i = 2;
      while d == d1
            i = i + 1;
           d = DistMedia(i-1,2);
           if d ~= d1
               break
           end
           DistsIguais(i-2,1) = DistMedia(i-1,1);
           DistsIguais(i-2,2) = d;
      end
mediasAval = calcularAvaliacaoMedia(turistas1,DistsIguais);
DistFinal(1:3,:) = mediasAval;
```

Por fim, listamos o 3 restaurantes mais similares ao escolhido inicialmente

```
listarRestaurantes(DistFinal(:,1),rest)
```

Opção 5

O utilizador insere o ID de um restaurante e, a aplicação indica uma estimativa do número de avaliações para o mesmo.
No primeiro script é logo inicializado o bloom filter e vai ser incrementado o número de vezes que cada restaurante foi avaliado.

```
function bloomFilter = addToBloomFilter(bloomFilter, key, k)
   % bloomfilter = array bloomfilter
   % k = number of hash functions
   % key = key to add
   for i = 1:k
       % aplicar a funçao de dispersao ao elemento
       h = hashFunction(key, i, length(bloomFilter));
       % incrementar a posição h do vetor -> filtro de Bloom com contagem
       bloomFilter(h) = bloomFilter(h) + 1;
   end
end
```

Depois é usada a função `checkBloomFilter` para saber o valor estimado de avaliações feitas ao restaurante inserido.

```
function nAval = checkBloomFilter(bloomFilter, key, k)
   contagens = zeros(1, k);
   for i = 1:k
       % aplicar a função de dispersão (hash) ao elemento
       h = hashFunction(key, i, length(bloomFilter));
       % contagem para cada função de hash
       contagens(i) = bloomFilter(h);
   end
   % a estimativa é o mínimo das contagens coletadas
   nAval = min(contagens);
end
```

E no fim o programa dá print da estimativa.

Relativamente à escolha do dimensionamento dos filtros de Bloom, usamos as fórmulas que estão nos slides teóricos e obtivemos estes resultados.

```
m = length(rest(:,1));
p = 0.01;
n = round(- (m * log(p)) / (log(2)^2));
k = round((n * log(2)) / m);
```

- m = length(restaurantes) = 213
- p = 0.01 -> representa a probabilidade máxima aceitável de um obter um falso positivo
- n = 2042
- k = 7

Conclusão

Com a realização deste trabalho, conseguimos consolidar os nossos conhecimentos nos tópicos abordados no guião 4, incluindo hash functions, algoritmo MinHash, filtros de Bloom e conceitos de similaridade e distância de Jaccard.

Este projeto, focado mais na componente prática, não só aprimorou as nossas habilidades com MATLAB, mas também desenvolveu a nossa capacidade de implementar algoritmos probabilísticos. Além disso, reforçou a importância do trabalho em equipa, permitindo-nos aplicar a teoria à prática de forma mais eficiente e colaborativa.