Universidade de Aveiro

P04

Tiago Portugal 103931

David Cobileac 102409

Métodos Probabilísticos para Engenharia Informática - 40337 1º Semestre DETI

30/01/2022

PL4

Neste guião foi nos dada a tarefa de criar uma interface para a consulta de uma base de dados composta por 2 ficheiros **friends.txt** e **users.txt**.

O ficheiro de amigos é composto por 3 colunas

- **ID de utilizador**
- **ID de um amigo**
- **Data de ultimo contacto**

sendo que um **ID** é repetido consoante o numero de amigos

O ficheiro de utilizadores é composto por no máximo 20 colunas sendo a primeira o **ID**, a segunda, o **Nome**, a terceira o **Segundo nome/nomes** seguidas por um numero aleatório, de 1 a 17, de colunas com os interesses desse utilizador.

Ex :

> 1;Matilde;Torres Moura;Música;Leitura;História;Andebol;Ténis

As opções a implementadas são as seguintes:

1. Your friends
2. Interests from similar user
3. Search name
4. Find most similar user based in list of interests
5. Exit

0. Pré processamento

A primeira coisa que o programa principal faz ao ser executado é fazer o pré processamento dos dados caso não tenha sido feito, procedendo á armazenação de várias variáveis a ser utilizadas pelas diversas funções ao longo do programa.

```
[users,friends,categorias,interesses_docs,interesses_ass,names_ass,names_shingl
es,hash_param,Bloom_filter,bloom_hashf] = Init()
```

- **Users**

Cell array onde são armazenados as 3 primeiras colunas dos **users**

- **friends**

Cell array onde são armzenadas as 3 (5 se separamos as datas pelo dia, mês e ano)

- **Categorias**

Cell array com as categorias retiradas dos utilizadores

- **interesse_docs_**

Array de documentos binários com #Categorias shingles onde índice **i** no documento a um 1 significa que o utilizador do índice **n** têm interesse na categoria **i**.

- **interesses_ass**

Assinaturas dos documentos feitos com uma **minHash** de 100 funções

- **names_shingles**

Shingles calculadas apartir dos nomes de todos os utilizadores

- **name_ass**

Assinaturas de documentos relativos aos nomes, que não chegamos a precisar e por isso não é armazenado.

- **hash_param**

Parâmetros da **minHash** usado para calcular as assinaturas dos nomes, que depois vamos utilizar para comparar utilizadores após o calculo das assinaturas.

- **Bloom filter**

Filtro bloom para verificar se um nome pertence a um utilizador na base de dados

- **bloom_hashf**

Conjunto de funções para usadas para adicionar e verificar se utilizadores pertencem ao filtro_bloom

1. Your friends

Nesta operação apenas percorremos o array de amigos até encontrar o **ID** do ultilizador e imprimimos os seus amigos.

```
fprintf("\n");
for i=1:length(friends)
    if(friends{i,1} == ID)
          fprintf("Id: %d Nome: %s
%s\n",friends{i,2},users{friends{i,2},2},users{friends{i,2},3});
              if(friends{i+1,1}~=ID)
                  break;
              end
    end
end
```

2. Interests from similar user

Aqui simplesmente recolhemos os amigos e as suas assinaturas de interesses, e como o grupo é pequeno simplesmente fazemos a **similiridade de Jacard diretamente**.

```
% Obter matriz dos interesses dos amigos de ID e seus IDs
[ids_amigos, interesses_amigos] = obterAmigos(ID, friends, interesses_docs);

Set = {};
for i=1:size(interesses_amigos,2)
    Set{end+1}=getInterestsStrings(interesses_amigos(:,i), categorias);
end

Nu = length(Set);
% Calcula a distaˆncia de Jaccard entre todos os pares pela definicnao.
J=zeros(1,Nu); % array para guardar distaˆncias
for n2= 2:Nu
    i=intersect(Set{1},Set{n2});
    u=union(Set{1},Set{n2});
    J(1,n2)=1-(length(i)/length(u));
end

[m,most_similar_user]=min(J(1,2:end));
most_similar_user = ids_amigos{most_similar_user+1};

fprintf("Amigo mais similar aos seus interesses:\nId: %d Nome: %s
%s\n\n",most_similar_user,users{most_similar_user,2},users{most_similar_user,3}
);

interesses_de_msu = interesses_docs{most_similar_user};
fprintf("Interesses de %d:\n",most_similar_user);
for i=1: length(categorias)
    if (interesses_de_msu(i)==1)
        fprintf("%s\n",categorias{i});
    end
end

interesses_de_msu = interesses_docs{most_similar_user};
fprintf("\nNovas sugestoes para %d:\n",ID);
for i=1: length(categorias)
    if (interesses_de_msu(i)==1 && interesses_docs{ID}(i)==0)
        fprintf("%s\n",categorias{i});
    end
end
```

3. Search name

Na terceira implementação queremos procurar um nome no meio de **mil** utilizadores pelo que para proceder a isto passamos por dois passos.

- Verificar a pertença do nome ao grupo de utilizadores com um **Bloom filter**
- Calcular a assinatura do nome com as **name_shingles** e a **minHash** com os os parametros **hash_param** e ver os utilizadores mais similares consoante as probabilidades obtidas.

```
name = input("Intruduza o nome completo de quem pretende procurar:\n\n");

name = name(find(~isspace(name)));

if(membro(name,Bloom_filter,bloom_hashf))
    doc = zeros(1,length(names_shingles));

    for j=1:length(name_shingles)
        doc(j) = contains(name,name_shingles{j});
    end

    doc = doc';
    ass_name = minHash(doc,hash_param);

    probs = zeros(1,1000);

    for i=1:1000
        prob(i) = sum(names_ass(:,i) == ass_name)/100;
    end

    candidates = sort(find(prob(i)>0.7 == 1),'descend');

    for i=1:length(candidates)
        u = user{candidates(i)};
        fprintf("ID %d: %s %s\n",u{1},u{2},u{3});
    end

else
    fprintf("Ultlizador não encontrado");
end
```

Os parametros do **Bloom filter** são 8000 bits, e **k** hash functions. Estes foram calculados assumindo um tamanho do filtro razoável e aplicando as seguintes formulas que derivam do estudo do **k** ótimo e da **probabilidade de falsos positivos**

$$k_{ótimo} = \frac{0.693n}{m}$$

$$p_{fp} \approx (1 - e^{-km/n})^k$$

4. Find most similar user based in list of interests

Nesta opcao e nos pedido para listar os amigos do utilizador atual escolhendo um desses amigos e apresentar os 3 utilizadores mais similares ao amigo escolhido com base nos interesses dele. Dividimos essa tarefa em 3 partes:

- Listar os amigos de ID e guardar o valor inserido pelo utilizador
- Calcular a matriz de assinaturas atraves do minHash
- Determinar os 3 utilizadores mais similares

```
for i=1:length(friends)
    if(friends{i,1} == ID)
        fprintf("Id: %d Nome: %s
%s\n",friends{i,2},users{friends{i,2},2},users{friends{i,2},3});
        if(friends{i+1,1}~=ID)
            break;
        end
    end
end

friend_id = input("\nchoose one of the friends:\n\n");

% Transpose interesses cell array
interesses_post = [];
for i=1:length(users)
    interesses_post = [interesses_post interesses{i}'];
end

% Obter matriz de assinaturas
signatures = minHash(interesses_post);
similar_users=findMostSimilarUsers(friend_id,signatures,3);

fprintf("\nUsers mais similares aos interesses de\nId: %d Nome: %s
%s\n====================\n",friend_id,users{friend_id,2},users{friend_id,3});

for i=1: length(similar_users)
    fprintf("\nId: %d Nome: %s
%s",similar_users(i),users{similar_users(i),2},users{similar_users(i),3});
end
```

A funcao minHash baseia-se na implementacao que esta no livro "Mining of Massive Datasets" (pg. 84):

```
function signatures = minHash (Set)

  nS = size(Set,1);  %Total ammount of Set elements
  nHash = 100;       %Total ammount of Hash Functions we'll use
  signatures = ones(nHash,size(Set,2))*999999999; % Each Row -> Hashing of That
Set's Entry; Each Col -> A Set entry

  h= waitbar(0,'Calculating');
  v = InitHashFunctions(100000,nHash);

  hcodes = zeros(nHash);
  for nu= 1:nS
      waitbar(nu/nS, h);

      % Calculamos h1(row),h2(row),...,hn(row)
      for nh= 1:nHash
          hcodes(nh)= mod(v.a(nh)*(nu)+v.b(nh),v.p);
      end

      fs = Set(nu,:);

      for nf= 1:length(fs)
          % Se row(col) tem um 0, nao fazemos nada
          if (fs(nf)==0)
              continue
          end

          % Porem, se row(col) tem um 1, entao para cada i=1,2,...,n
signatures(i,col) para o menor valor entre a entrada
          % na matriz de assinaturas e hi(row)
          for nh= 1:nHash
              if (hcodes(nh)<signatures(nh,nf))
                  signatures(nh,nf)=hcodes(nh);
              end
          end
      end
  end
  delete (h)
end
```

5. Conclusão

Com este guião ganhamos uma maior visão da importância da componente probabilística no desenvolvimento de software podendo fazer com que cálculos computacionalmente intensivos

sejam feitos com uma visivelmente mais eficientemente como no calculo da similaridade.