deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

PL 4
Algoritmos Probabilísticos

Métodos Probabilísticos para Engenharia Informática

Professor Carlos Bastos e Professor António Teixeira

Trabalho realizado por:

João Nuno da Silva Luís, nº 107403, P1

Diana Raquel Rodrigues Miranda, nº 107457, P1

08-01-2023

Índice

Introdução

Este trabalho é realizado no âmbito da disciplina de Métodos Probabilísticos para Engenharia Informática do 2º ano da Licenciatura em Engenharia Informática.

Foi-nos proposto desenvolver uma aplicação, em MATLAB, com algumas funcionalidades de um sistema online de disponibilização de filmes. Esta aplicação vai fazer uso de 3 ficheiros previamente disponibilizados. O primeiro ficheiro é o "u.data", de onde iremos retirar a informação necessária para identificar os utilizadores do sistema, os filmes que cada utilizador viu e a avaliação atribuída por cada utilizador. O segundo ficheiro será o "users.txt", que nos vai dar as informações sobre cada utilizador. E por último, temos o ficheiro "film_info.txt" que nos vai fornecer as informações sobre cada filme disponível.

Com isto em mente, temos como objetivo principal realizar uma aplicação totalmente funcional, com todas as opções do menu operacionais.

Desenvolvimento da Aplicação

Numa primeira fase, criámos dois scripts, um para armazenar todas as estruturas de dados associadas aos utilizadores e aos filmes – **script_data.m** – e outro para implementar a aplicação em si – **script_app.m**. Esta separação da informação simplifica a execução da aplicação, uma vez que facilita o carregamento dos dados necessários para o script da aplicação.

script_data.m

No ficheiro script_data.m começámos por construir um set que armazena o conjunto de utilizadores por filme e a respetiva avaliação de cada um, através da função "create_structure.m" criada na aula prática. Em seguida, criámos o cell array "dic" e "dic2", o primeiro irá guardar as informações de cada utilizador e outro as informações sobre os filmes, respetivamente.

```
[Set, films] = create_structure('u.data');

dic = readcell('users.txt', 'Delimiter', ';');
nUsers = height(dic);

dic2 = readcell('film_info.txt', 'Delimiter', '\t');
nFilms = height(dic2);
```

Figura 1 - Inicialização do ficheiro script_data.m

```
function [Set, films] = create_structure(ficheiro)
udata=load(ficheiro); % Carrega o ficheiro dos dados dos filmes
u = udata(1:end,1:3); clear udata;

% Lista de utilizadores
films = unique(u(:,2));
nFilms= length(films);

Set= cell(nFilms,1);

for n = 1:nFilms
    ind = find(u(:,2) == films(n));
    Set{n} = [Set{n} u(ind,1) u(ind,3)];
end
end
```

Figura 2 - Função create_structure

Em seguida calculámos as matrizes de assinaturas com os vetores *MinHash* para as diferentes opções, para isso, desenvolvemos as funções em ficheiros à parte, de maneira obter um código mais simples e limpo.

Em primeiro, relativamente às assinaturas correspondentes ao conjunto de utilizadores que avaliaram cada filme, usámos a função para calcular o *MinHash* criada na aula para a resolução da secção 4.3 do guião PL4, fazendo apenas algumas alterações para adaptar a função ao que nos é pedido na opção 1. A função calcula uma "assinatura" para cada conjunto no *Set* aplicando uma função hash (DJB31MA_Modified) a cada item do *set* e retirando o mínimo de todos os valores hash resultantes, que correspondem aos melhores valores obtidos dentro das 1000 *hash functions*. As assinaturas resultantes são devolvidas numa matriz.

```
% Cálculo do MinHash correspondente aos utilizadores
nHF = 1000;
MinHashUsers = MinHash(Set,nHF);
```

Figura 3 - Código do ficheiro sricpt_data.m

```
function signatures = MinHash(Set,numHash)
nfilms = length(Set);
signatures = Inf(nfilms,numHash);
h = waitbar(0,'Calculating MinHash');
tic
for i = 1:nfilms
    waitbar(i/nfilms,h);
    nUsers = length(Set{i});
    for j = 1:nUsers
        key = num2str(Set{i}(j));
        hash = DJB31MA_Modified(key,127,numHash);
        signatures(i,:) = min(hash,signatures(i,:));
    end
end
delete (h)
end
```

Figura 4 - Função usada para cálculo do MinHash

Em segundo, relativamente às assinaturas correspondentes ao conjunto de interesses de cada utilizador, começámos por retirá-los do *cell array* **dic**, através da função *getInterests*. Esta função extrai a lista de interesses e uma matriz de assinaturas correspondente do *cell array* com os dados dos utilizadores. A lista de interesses é criada através de um loop por cada utilizador existente extraindo todos os interesses listados. A matriz de assinaturas é criada fazendo um loop por cada interesse e por cada utilizador, e definindo a entrada correspondente na matriz para 1, se o utilizador que está a ser avaliado tiver esse interesse. A função retorna tanto a lista de interesses como a matriz de assinatura. Com estes dados procedemos, em seguida, ao cálculo do *MinHash*, com a utilização da função *MinHash_Inte*. Esta função percorre cada utilizador e extrai os índices dos interesses que o utilizador tem. Para cada um destes interesses, a função calcula um conjunto de valores hash utilizando uma função hash (DJB31MA) e o nome do interesse como entrada. O mínimo destes valores de hash é então armazenado na linha correspondente de **MinHashInterests**.

```
% Cálculo do MinHash correspondente aos interesses
nHF = 200;
[Interests, sigInterests] = getInterests(nUsers,dic);

%MinHash Interesses
MinHashInterests = MinHash_Inte(sigInterests,Interests, nHF, nUsers);
```

Figura 7 - Código do ficheiro sricpt_data.m

```
function MinHashInterests = MinHash_Inte(sigInterests,Interests, nHF, nUsers)
MinHashInterests = inf(nUsers, nHF);
x = waitbar(0,'A calcular MinHashInterests()...');
for k= 1 : nUsers
    waitbar(k/nUsers,x);
    interestsIdx = find(sigInterests(:, k));
    for j = 1:length(interestsIdx)
        chave = char(Interests{interestsIdx(j)});
        for i = 1:nHF
            chave = [chave num2str(i)];
            h(i) = DJB31MA(chave, 127);
        end
        MinHashInterests(k, :) = min([MinHashInterests(k, :); h]);
    end
end
delete(x);
end
```

Figura 5 - Função usada para cálculo do MinHash

```
function [Interests, sigInterests] = getInterests(nUsers,users)

Interests = {};
k = 1;

for i= 1:nUsers
    for j= 4:17
        if ~anymissing(users{i,j})
            Interests{k} = users{i,j};
            k = k+1;
        end
    end
end

Interests = unique(Interests);

nInterests = length(Interests);
sigInterests = zeros(nInterests,height(users)); %matriz de assinaturas

for i= 1:nInterests
    for n= 1:nUsers
        for k= 4:17
            if ~anymissing(users{n,k})
                if strcmp(Interests(i),users{n,k})
                    sigInterests(i,n) = 1;
                end
            end
        end
    end
end
```

Figura 6 - Função usada para extrair os interesses dos utilizadores

Em terceiro, calculámos o *MinHash* para os títulos dos filmes. Para isso, usámos 200 hash functions e atribuímos à variável **shingles** o valor 4. Este valor foi definido depois de experimentar os valores entre 2 e 5 e perceber que o valor 4 é o que oferece um resultado mais fiável. A forma como calculámos o *MinHash* neste caso difere da forma como calculámos para os utilizadores e para os interesses, pois neste caso usámos os shingles para trabalhar com os títulos.

```
% Cálculo do MinHash correspondente aos Titulos
nHF = 200;
shingles = 4;
MinHashTitles = MinHashStrings(nFilms, dic2, shingles,nHF);
```

Figura 8 - Código do ficheiro sricpt_data.m

```
function MinHashString = MinHashStrings(n,dic,shingles,hf)

MinHashString=inf(n,hf);

x = waitbar(0,'Calculating MinHashString');
for n1=1:n
    waitbar(n1/n,x);
    string=lower(dic{n1,1});
    for n2=1:length(string)-shingles+1
        shingle=string(n2:n2+shingles-1);
        h=zeros(1,hf);
        for i=1:hf
            shingle=[shingle num2str(i)];
            h(i)=DJB31MA(shingle,127);
        end
        MinHashString(n1,:)=min([MinHashString(n1,:);h]);
    end
end
delete(x);
end
```

Figura 9 - Função usada para cálculo do MinHash

Depois de todas as matrizes com os vetores *MinHash* necessárias, iniciámos o *Bloom Filter*. Começamos por definir o tamanho do filtro, fazendo algumas experiências com vários números, e percebemos que um número menor que 10000 não dava um resultado tão fiável e um número maior que 10000 não alterava muito os resultados, pelo que definimos o tamanho como 10000. Em seguida, calculámos o valor ideal para **k** e fizemos a inicialização do filtro de Bloom. Após isto, adicionámos os elementos ao filtro. Esses elementos vão ser todas as avaliações maiores ou iguais a 3 dadas por cada utilizador a cada filme.

```
% Bloom Filter
n = 10000;
k =  round(n*log(2)/nFilms);
countingBF = Inicializar_FiltroBloom(n);

for i=1:nFilms
    array = Set{i};
    array(array(:,2) < 3,:) = []; %remover users que deram menos de 3
    for x = 1:length(array(:,2))
        countingBF = Adicionar_FiltroBloom(countingBF,i,k,n);
    end
end
```

Figura 10 - Código do ficheiro sricpt_data.m

Por fim, guardámos na estrutura "data.mat" todas as variáveis para a execução da aplicação.

```
save 'data.mat' Set dic nUsers dic2 nFilms MinHashUsers MinHashTitles MinHashInterests countingBF,
```

Figura 11 - Código do ficheiro sricpt_data.m

script_app.m

No ficheiro "script_app.m", começamos por importar todos os dados contidos no ficheiro "data.mat", dados esses calculados no "script_data.m".

Menu

Ao executar a aplicação, esta começa por pedir o ID do filme que se torna o filme atual, sendo que é verificado se é inserido um ID válido, ou seja, um número entre 1 e 1682.

Após este passo, e para interagir com o utilizador, usámos a função *menu()*, própria do MATLAB. Esta exibe as 4 opções a desenvolver, sendo que após o utilizador escolher uma das opções e a aplicação a executar, vai ser mostrado novamente o menu. Assim, as únicas formas de terminar a aplicação são através do botão "5- Exit" ou do botão "Stop", este último próprio do MATLAB.

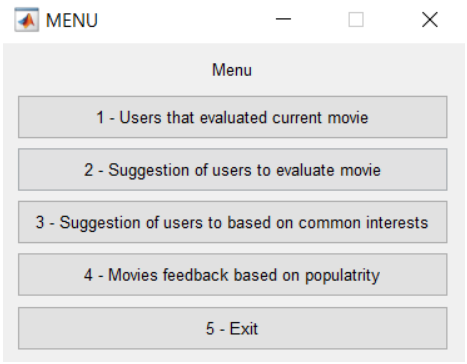

Figura 12- Menu()

Opções Desenvolvidas

Opção 1

Na opção 1, a aplicação vai listar os ID's e os respetivos nomes, um por linha, dos utilizadores que avaliaram o filme atual.

Assim, vamos ao *cell array* **Set**, que contém os ID's dos utilizadores que viram cada filme, e vamos guardar na variável **userIDs,** os ID's que viram o filme em questão. De seguida, e com recurso a um *for* loop, vai ser impresso no *Command Window* do MATLAB o ID do utilizador, e o nome próprio e apelidos, ambos com recurso à variável **dic**, que contém todos os dados dos utilizadores listados no ficheiro fornecido *users.txt*.

```
case 1
    fprintf('\nUsers who rate the movie "%s":\n',dic2{filmID});
    userIDs = Set{filmID};
    for i=1:length(userIDs)
        fprintf(" (ID: %d) %s %s\n", userIDs(i), dic{userIDs(i),2}, dic{userIDs(i),3})
    end
```

Figura 13 - Execução da primeira opção

Opção 2

Nesta opção, a aplicação determina os 2 filmes mais similares ao filme atual, tendo por base os utilizadores que avaliaram cada um dos filmes. De seguida, é apresentado os utilizadores que avaliaram pelo menos um dos filmes mais idênticos, mas com a restrição de estes ainda não terem avaliado o filme atual.

Assim, começámos por calcular as distâncias de Jaccard entre os filmes, através da matriz de assinaturas com os vetores MinHash correspondente ao conjunto de utilizadores, calculado anteriormente

Após este passo, utilizámos a função *sort()* para ordenar a nossa matriz **distance** por ordem crescente, podendo assim obter os ID's dos filmes mais similares.

```
fprintf("\nOption 2 - \n")
nHF = 1000;
distance = zeros(nFilms, 1);
for i = 1:nFilms
    if i ~= filmID
        distance(i)=1-sum(MinHashUsers(filmID,:)==MinHashUsers(i,:))/nHF;
    else
        distance(i) = 1;
    end
end

% ordena o array por ordem crescente
[~, idx] = sort(distance, 'ascend');
mostSimilar = idx(1:2); % vai buscar os dois primeiros que são os mais similares
fprintf('The two most similar films to film %d are %d and %d\n', filmID, mostSimilar);
```

Figura 14-Distâncias de jaccard e filmes mais similares

Depois, fomos encontrar os utilizadores que tivessem avaliado pelo menos um desses filmes, tendo utilizado a função *setdiff()* para ignorar os utilizadores que já tinham avaliado o filme atual.

Para terminar, imprimimos os utilizadores que satisfizeram estas condições, tendo o cuidado de anteriormente utilizar a função *unique()*, para que não fossem impressos utilizadores repetidos. A forma de imprimir esses mesmos utilizadores foi igual à da opção 1.

```
% Encontra os utilizadores que avalariam pelo menos um dos filmes similares
usersInfo = [];
for i = 1 : length(mostSimilar)
    x = mostSimilar(i);
    c = setdiff(Set{x,:}, Set{filmID,:})';
    usersInfo = [usersInfo, c];
end

usersInfo = unique(usersInfo);

fprintf('Users who rated one of the similar movies, but not the movie "%s":\n', dic2{filmID});
for i = usersInfo
    fprintf(" (ID: %d) %s %s\n", i, dic{i,2}, dic{i,3})
end
```

Figura 15 - Continuação da execução da opção 2

Opção 3

Nesta escolha, a aplicação seleciona os utilizadores cuja distância de Jaccard estimada (em termos de interesses) é menor que 0.9 e que ainda não tenham visto o filme atual. Depois, apresenta no ecrã os ID's e os nomes dos 2 utilizadores que aparecem no maior número de conjuntos.

Com isto, começámos por calcular as distâncias e guardar numa variável **conjunto** o conjunto (**n1** e **n2**) e a distância correspondente, caso esta fosse menor que 0.9. Como para alguns filmes esta ação demorava algum tempo, decidimos colocar um aviso para indicar que os dados estavam a ser carregados.

```
fprintf("\nOption 3 - \n")
userIDs = Set{filmID};
threshold=0.9; % limiar da decisão para a Dist. de Jaccard
nHF = 200;
conjunto = zeros(nUsers,3);
count = 1;
disp('Loading data, please wait...')
for i= 1:length(userIDs)
    n1 = userIDs(i);
    for n2=1:nUsers
        if ~ismember(n2, userIDs(:,1))
            distanceInterests = 1-sum(MinHashInterests(n1,:)==MinHashInterests(n2,:))/nHF;
            if distanceInterests<threshold
                conjunto(count, 1) = n1;
                conjunto(count, 2) = n2;
                conjunto(count, 3) = distanceInterests;
                count = count + 1;
            end
        end
    end
```

Figura 16 - Cálculo da distância de Jaccard na opção 3

Em seguida, fizemos a contagem da quantidade de vezes que cada utilizador aparece nos conjuntos. Para isso, através do comando *unique()* do MATLAB guardámos numa variável todos os utilizadores que apareciam nos conjuntos. Depois para cada um desses utilizadores contámos quantas vezes apareciam e guardámos a menor distância que esse utilizador teve com algum dos outros utilizadores, guardando essa informação na variável **counts**.

```
conjuntoUnique = unique(conjunto(:,2));
counts = zeros(length(conjuntoUnique),2);
for i=1:length(conjuntoUnique)
    u = conjuntoUnique(i);
    count = 0;
    dist = 1;
    for x=1:length(conjunto(:,2))
        c = conjunto(x,2);
        if c == u
            count = count +1;
            if dist > conjunto(x,3)
                dist = conjunto(x,3); % guardar a menor distância
            end
        end
    end
    counts(i,1) = u;
    counts(i,2) = count;
    counts(i,3) = dist;

end
```

Figura 17 - Contagem que quantas vezes cada utilizador aparece nos conjuntos (opção 3).

Por fim, ordenamos a variável **counts** por ordem decrescente da segunda coluna (nº de vezes que cada utilizador aparece) e pela ordem crescente da terceira coluna (distância mínima de cada utilizador). Ordenámos desta maneira para que quando houvesse um empate no número de vezes que o utilizador aparece, fosse impresso no ecrã o que tinha uma distância menor. Para apresentar a informação desejada, fomos buscar os dois primeiros valores da variável **counts** e fizemos a impressão no ecrã.

```
counts = sortrows(counts, [-2, 3]);

mostCommon = counts(1:2);
fprintf('The two users who appear in more sets are:\n');
for i = mostCommon
    fprintf(" (ID: %d) %s %s\n", i, dic{i,2}, dic{i,3})
end
```

Figura 18 - Print da informação desejada (opção 3).

Opção 4

Aqui, a aplicação pede ao utilizador que insira uma *string*, e a partir dessa são devolvidos os 3 filmes com os títulos mais similares à *string* introduzida, e para cada nome, o número de vezes que o filme foi avaliado com uma nota superior ou igual a 3.

Assim, começámos por fazer o cálculo da distância entre os filmes e a *string* inserida, adaptando a função *MinHashString* utilizada anteriormente no cálculo dos *MinHash* para os títulos dos filmes. A alteração consiste em passar como parâmetro um uma *string* em vez de um *cell array* (**dic**). Para este cálculo foram usadas 100 *hash functions* e o *shingle* com tamanho 4.

```
fprintf("\nOption 4 - \n")
string = lower(input('Write a string: ','s'));
shingle = 4;
nHF = 200;
MinHashString = Function_MinHashString(string, shingle,nHF);
distanceFilms = zeros(nFilms, 1);
for i = 1:nFilms
    distanceFilms(i)=1-sum(MinHashString(1,:)==MinHashTitles(i,:))/nHF;
end
```

Figura 19 - Cálculo distância de Jaccard (opção 4).

```
function MinHashString = Function_MinHashString(string, shingle,nhf)
    MinHashString=inf(1,nhf);
    for j=1:length(string)-shingle+1
        shingles=string(j:j+shingle-1);
        h=zeros(1,nhf);
        for m=1:nhf
            shingles=[shingles num2str(m)];
            h(m)=DJB31MA(shingles,127);
        end
        MinHashString(1,:)=min(MinHashString(1,:),h);
    end
end
```

Figura 20 - Função MinHash (opção 4).

De seguida, ordenámos a variável **distanceFilms**, que armazena todas as distâncias calculadas, de forma crescente para retirar os primeiros 3 valores que correspondem aos 3 filmes mais similares. Depois, procedemos à contagem de quantas vezes esses 3 filmes foram avaliados com nota superior ou igual a 3, recorrendo ao filtro calculado anteriormente, na execução do "script_data.m".

```
[~, idx] = sort(distanceFilms, 'ascend');

fprintf("Suggested Movies: \n")
mostSimilar = idx(1:3);
n = 10000;
k = round(n*log(2)/nFilms);

for i=1:length(mostSimilar)
    x=mostSimilar(i);
    count = Membro_FiltroBloom(countingBF,mostSimilar(i),k,n);
    fprintf(" (ID: %d) %s -  Nº. of times it was rated >= 3: %d \n", x, dic2{x}.count)
```

Figura 21 - Contagem com Filtro de Bloom (opção 4).

Considerações tomadas na implementação

Nesta secção, reforçamos a escolha de alguns valores de certas variáveis, relacionadas com a implementação dos métodos probabilísticos.

Para decidir qual o melhor número de *hash functions* a utilizar para o cálculo da matriz de assinaturas com os vetores *MinHash* correspondentes ao conjunto de interesses, testámos alguns valores e verificámos os resultados obtidos para alguns ID's fixos, que neste caso serão 1, 545 e 1005 ao selecionarmos a terceira opção.

	ID: 1	**ID: 545**	**ID: 1005**
nHF = 50	ID: 214 - 445 vezes ID: 439 - 442 vezes	ID: 356 - 12 vezes ID: 787 - 12 vezes	ID: 609 - 22 vezes ID: 75 - 22 vezes
nHF = 100	ID: 214 - 446 vezes ID: 571 - 445 vezes	ID: 787 - 12 vezes ID: 356 - 12 vezes	ID: 284 - 22 vezes ID: 504 - 22 vezes
nHF = 200	ID: 214 - 447 vezes ID: 439 - 446 vezes	ID: 787 - 12 vezes ID: 356 - 12 vezes	ID: 261- 22 vezes ID: 504 - 22 vezes

Figura 22 - Resultado da opção 3 utilizando vários valores de hash functions.

Ao analisar a tabela acima, percebemos que os ID's do resultado são semelhantes apesar do valor de *hash functions* serem diferentes. Com isto, concluímos então que o melhor valor de *hash functions* a serem utilizadas é 200.

Após o número de *hash functions* definido, fomos determinar qual o melhor valor a ser atribuído à variável **shingles**, que simboliza o tamanho de cada *shingle*.

	String = home al	**String = story**	**String = furious**
shingles = 2	(ID: 94) Home Alone (1990) - 95 (ID: 894) Home Alone 3 (1997) - 5 (ID: 304) Fly Away Home (1996) - 132	(ID: 1) Toy Story (1995) - 417 (ID: 1105) Firestorm (1998) - 10 (ID: 305) Ice Storm, The (1997) - 73	(ID: 489) Notorious (1946) - 51 (ID: 30) Belle de jour (1967) - 35 (ID: 1437) House Party 3 (1994) - 2
shingles = 3	(ID: 94) Home Alone (1990) - 95 (ID: 894) Home Alone 3 (1997) - 5 (ID: 304) Fly Away Home (1996) - 132	(ID: 1) Toy Story (1995) - 417 (ID: 1105) Firestorm (1998) - 10 (ID: 277) Restoration (1995) - 59	(ID: 489) Notorious (1946) - 51 (ID: 1437) House Party 3 (1994) - 2 (ID: 366) Dangerous Minds (1995) - 37
shingles = 4	(ID: 94) Home Alone (1990) - 95 (ID: 894) Home Alone 3 (1997) - 5 (ID: 304) Fly Away Home (1996) - 132	(ID: 1) Toy Story (1995) - 417 (ID: 308) FairyTale: A True Story (1997) - 20 (ID: 1344) Story of Xinghua, The (1993) - 4	(ID: 489) Notorious (1946) - 51 (ID: 1113) Mrs. Parker and the Vicious Circle (1994) - 18 (ID: 1) Toy Story (1995) - 417
shingles = 5	(ID: 94) Home Alone (1990) - 95 (ID: 894) Home Alone 3 (1997) - 5 (ID: 304) Fly Away Home (1996) - 132	(ID: 1) Toy Story (1995) - 417 (ID: 478) Philadelphia Story, The (1940) - 99 (ID: 1344) Story of Xinghua, The (1993) - 4	(ID: 489) Notorious (1946) - 51 (ID: 1) Toy Story (1995) - 417 (ID: 2) GoldenEye (1995) - 106

Figura 23 - Resultado da opção 4 para diferentes strings e diferentes tamanhos de shingles.

Analisando a tabela, da figura 23, concluímos que o melhor valor para as *shingles* é 4, pois sugere-nos alguns filmes sem restringir tanto as opções como no caso de *shingles* = 5 nem oferece resultados não fidedignos como para *shingles* = 2 ou 3.

Conclusão

Com este trabalho, solidificámos os nossos conhecimentos acerca de funções de dispersão, filtros de Bloom e algoritmos de *MinHash*. Para além disso, aprendemos, numa vertente prática, a usar *shingles* e a usar noções de distâncias de Jaccard diferentes da abordagem "clássica".

Para além disto, aprofundámos também os nossos conhecimentos de MATLAB, já que ficámos a conhecer novas funções que o mesmo disponibiliza, assim como consolidámos a implementação das nossas próprias funções, numa abordagem de clarificar o código por nós implementado.

Por fim, concluímos que atingimos os objetivos propostos pelo guião, com todas as funcionalidades da aplicação totalmente implementadas.