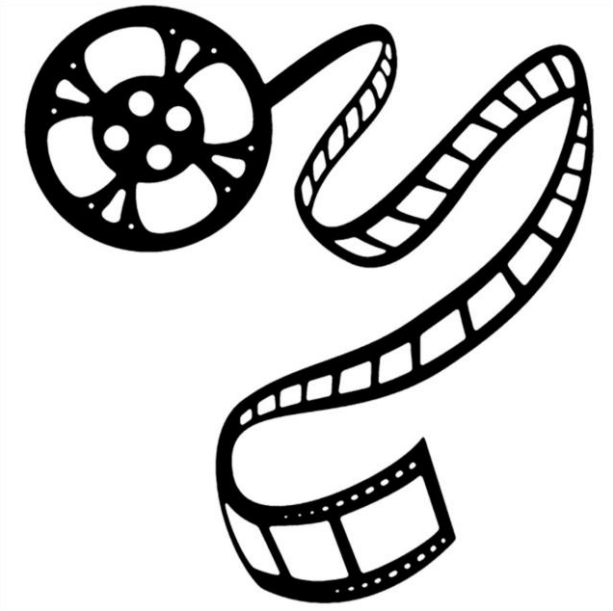
universidade de aveiro

Métodos Probabilísticos para Engenharia Informática

Trabalho Prático 3

Similaridade e Algoritmo *MinHash*



28.01.2021

- João Farias (98679)
- Artur Romão (98470)

Índice

Introdução

Este relatório foi realizado no âmbito do guião P04 da disciplina de MPEI. O exercício consiste em pedir um número que identifica um utilizador (*id*) e através desse número o programa permite executar três tarefas fundamentais diferentes:

1. Listar os filmes vistos por esse utilizador;
2. Pedir sugestões de filmes;
3. Pesquisar um filme através do título.

   Em cada uma destas opções vamos explicar como o fizemos.

Implementação

Tal como pedido, criamos dois ficheiros *matlab*: um contém os cálculos de *MinHashs* e a implementação da lista de filmes para cada utilizador e o outro ficheiro contém a *interface* da aplicação, bem como mais alguns cálculos necessários para resolver o problema pedido.

Opção 1

Na opção número 1 é pedido para listar os filmes de um certo utilizador. Para isso, começamos por, no ficheiro *ex1.m* (ficheiro complementar), carregar os ficheiros '*u.data*' e '*u_item.txt*' para as variáveis *uData* e *dic* (este ficheiro foi carregado como um *cell array*), respetivamente. No primeiro ficheiro os dados de cada coluna estão separados por tabs e A linha número **n** contém a informação do filme com o *id* **n** usado na segunda coluna do ficheiro *u.data*. A primeira coluna contém o nome do filme e as colunas de 2 a 20 contêm 1 ou 0 consoante o filme é classificado segundo um (ou mais) dos seguintes géneros:

- *unknown*
- *action*
- *adventure*
- *animation*
- *children*
- *comedy*
- *crime*
- *documentary*
- *drama*
- *western*
- *fantasy*
- *film-noir*
- *horror*
- *musical*

- *mystery*
- *romance*
- *sci-fi*
- *thriller*
- *war*

```
uData=load('u.data');
dic= readcell('u_item.txt');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% OPCÃO 1
utilizadores=uData(:,1);
filmes=uData(:,2); %id filmes
```

Fig. 1 - Variáveis essenciais

Por outro lado, o ficheiro *u.data*, tem na primeira coluna informação sobre os utilizadores e na segunda coluna informação sobre os filmes, sendo que o *id* do filme que aparece na mesma linha que o *id* do utilizador, significa que esse utilizador viu esse filme. Tendo isto em conta, torna-se bastante mais simples de implementar a opção número 1. Começamos por criar um vetor (*utilizadores*) com todos os dados da primeira coluna do ficheiro *u.data* (*id* de utilizadores) e outra variável *filmes* com informação sobre a segunda coluna do mesmo ficheiro, mas esta guarda informação dos *id* dos filmes. Como o número de filmes vistos por cada utilizador não é igual para todos, criamos um *cell array* com uma coluna e 943 linhas, em que cada linha **n** tem informação sobre o utilizador com o *id* **n** e, seguidamente, fizemos um ciclo *for* a percorrer os *id*'s de todos os utilizadores, em que numa parte inicial descobrimos os índices do *array* utilizadores, onde o valor do ciclo *for* (*i*) era igual aos valores armazenados nesse *array*. Depois disso armazenamos os filmes no *cell array* criado, através dos índices calculados anteriormente.

```
list=cell(943,1);
for i=1:max(utilizadores)
   ind=find(utilizadores==i);
   list{i,1}=dic(filmes(ind),1);
end
```

Fig. 2 - Cell array com os filmes visualizados por cada utilizador

No final deste processo, temos todos os filmes que cada utilizador viu. Quando quisermos imprimir a lista de filmes de um dado utilizador, basta ir à posição deste utilizador (*id*), no *cell array* "list", e imprimir a informação dessa célula.

```
case 1
    fprintf('%20s \n','Your Movies');
    disp(string(list{n,1}));
```

Fig. 3 - Código da interface para a primeira opção

Opção 2

Para a opção 2 é pedido para encontrar sugestões de filmes de acordo com o género escolhido pelo utilizador. Para isso, é necessário encontrar o *id* da pessoa mais parecida com o utilizador e apresentar todos os filmes deste género que essa pessoa viu, exceto os que o utilizador já tenha visto. Para isso, começamos por calcular o *minhash* dos utilizadores para 100 funções de *hash*, usando como base a função desenvolvida pelo professor nas aulas práticas ajustando apenas os valores usados. O cálculo deste *minhash* foi feito no ficheiro *ex1.m* para poupar tempo de execução no programa principal.

```
nUtilizadores=length(unique(utilizadores));
J=zeros(nUtilizadores);
K=100; %Funções de Hash
```

Fig. 4 - Inicialização de variáveis

Para calcular os *MinHash*, começamos por inicializar a matriz *MinHashTable*, de infinitos, com as seguintes dimensões: número de linhas igual ao número de utilizadores e número de colunas igual ao número de funções de *hash* usadas (usou-se o valor infinito como *default*, uma vez que no preenchimento desta *HashTable* vão sendo escolhidos os menores valores, por isso, o objetivo é começar com os maiores valores possíveis). Numa fase seguinte, fizemos um ciclo *for* de 1 até ao número máximo de utilizadores. Dentro desse ciclo *for*, é escolhido um conjunto de filmes associado a um utilizador e, dentro doutro ciclo *for* que itera sobre os títulos dos filmes, é calculada a *chave* que corresponde a cada caracter do nome do filme. É feito um último ciclo *for* que percorre o número de funções de *hash* usadas e que adiciona à *chave*, o número da iteração do último ciclo *for*. É de notar que esse número necessita de ser passado para o formato *string* usando para isso a função *num2str*. Ainda dentro do último ciclo *for*, é armazenado no vetor ***h***, o valor correspondente à mapeação desta informação através de *MinHash*. No final deste ciclo *for* esse valor é armazenado na matriz *MinHashTable*, apenas se for menor do que o lá guardado anteriormente.

```
MinHashTable=inf(nUtilizadores,K);
for i=1:nUtilizadores
    conjunto=list{i};
    for j=1:length(conjunto)
        chave=char(conjunto(j));
        for kk=1:K
            chave = [chave num2str(kk)];
            h(kk) = DJB31MA(chave, 127);
        end
        MinHashTable(i,:)=min([MinHashTable(i,:);h]);
    end
end
```

Fig. 5 - Preenchimento da MinHashTable

Por sua vez, no programa principal, e após ter escolhido a categoria de filmes pretendida (guardada na variável *categoria*), calculamos a distância de *Jaccard* fazendo para isso um ciclo *for* começando em 1 e acabando no *id* do utilizador mais alto e armazenamos no vetor *J* (na posição do índice desse ciclo) a distância correspondente (essa distância corresponde à distância entre o utilizador escolhido inicialmente e todas as outras pessoas). Dentro do ciclo *for*, somamos a linha **n** (correspondente ao *id* do utilizador) com a linha **n2 (**de notar que cada valor numa linha só é somado se o valor correspondente na segunda linha for diferente) correspondente ao índice do ciclo *for*, e no final dividimos pelo número de funções de hash.

```
J=zeros(nUtilizadores,1);
for n2=1:nUtilizadores
    J(n2)=sum(MinHashTable(n,:)~=MinHashTable(n2,:))/K;
end
```

Fig. 6 - Cálculo das distâncias de Jaccard

Tendo já as distâncias de *Jaccard* calculadas, precisamos de saber o índice onde ocorre a menor distância, ou seja, qual o utilizador mais parecido com o escolhido inicialmente. Para isso, usamos a função *mink* para determinar os dois menores valores, embora queiramos apenas um. O menor valor de todos é sempre 0, que corresponde ao *id* do utilizador, o que não nos interessa para o nosso caso. O segundo menor valor é, assim, o que pretendemos encontrar.

Para descobrir o índice onde ocorre o menor valor, usamos a função *find* e, caso esta devolva mais que um valor (pode acontecer caso haja mais que um utilizador com a mesma distância de *Jaccard*), escolhemos o primeiro índice. Depois disso, recorremos à função *string* para colocar num vetor, *filmesUtilizador2*, os filmes visualizados pelo utilizador cujo *id* acabamos de calcular.

De seguida, necessitamos de listar num vetor os filmes contidos na variável *dic* que pertencem à categoria selecionada anteriormente. Para tal, recorremos à função *cell2mat* para transformar os valores (em forma de célula) para um vetor, *isCategory*, que vai guardar os dados referentes à coluna da categoria (zeros e uns, em que zero significa que o filme não é desse género e um o contrário). Depois, executamos a seguinte linha de código: "*filmesCategoria = string(dic(find(isCategory == 1), 1));*", assim, a variável *filmesCategoria* irá receber todos os filmes do *cell array* que efetivamente fazem parte da categoria selecionada. Fazemos *find(isCategory == 1)* para obter os índices dos filmes. Após isso, e com o auxílio da função *string*, guardamos os títulos desses filmes no vetor, ao selecionar a primeira coluna da variável *dic*.

```
ind=mink(J,2);
ind=ind(2);

id=find(J==ind); %utilizador mais parecido com o escolhido (n)
id=id(1);

filmesUtilizador2=string(list{id,1}); % filmes do utilizador id
isCategory = cell2mat(dic(:,categoria+2)); % 0's e 1's (indica se o filme é ou não daquela categoria)
filmesCategoria=string(dic(find(isCategory==1),1)); % Todos os filmes da categoria escolhida

suggestions = string(1);
indice = 1;
```

Fig. 7 - Determinação do utilizador mais semelhante, dos seus filmes e listagem dos filmes da categoria pretendida

Já com os filmes da categoria todos guardados, é agora necessário concluir quais desses filmes foram visualizados pelo utilizador. Para tal, começamos por inicializar duas variáveis: um *array* de strings, *suggestions* e um contador, *indice*. De seguida, fazemos um ciclo *for* que itera sobre os filmes visualizados pelo utilizador e, recorrendo a uma condição *if* e à função *ismember*, verificamos se esse filme pertence à dita categoria e se o nosso primeiro utilizador ainda não o viu. Se tal se confirmar, o filme é adicionado ao *array suggestions* e o contador é incrementado.

Tendo por fim o *array suggestions* preenchido, temos tudo para apresentar as sugestões de filmes. Fazemos uma condição *if* para concluir se o *array* foi, de facto, preenchido por títulos de filmes. Se tal se verificar, é feito um ciclo *for* que imprime esses mesmos títulos, caso contrário, é impressa a seguinte mensagem: "*Cannot find any films*".

```
for filme=1:length(filmesUtilizador2)
    if (ismember(filmesUtilizador2(filme), filmesCategoria) && ~ismember(filmesUtilizador2(filme),filmesUtilizador1))
        suggestions(indice) = filmesUtilizador2(filme);  % na condição de cima, verifica-se se o filme pertence ao género
        indice = indice + 1;                    % selecionado e se o utilizador1 ainda não o viu
    end
end
if (suggestions(1)~='1')
    fprintf('\n');
    fprintf("%40s\n", 'Suggetions');
    for i=1:length(suggestions)
        fprintf("%10s%s\n","- ",suggestions(i));
    end
    fprintf("\n");
else
    fprintf("\n");
    fprintf("%10s %s\n", '-', 'Cannot find any films');
    fprintf("\n");
end
```

Fig. 8 - Determinação e apresentação das sugestões

Opção 3

Na opção 3, também à semelhança do que fizemos na opção 2, começamos por calcular os *MinHash*, desta vez, dos títulos dos filmes, usando, novamente, uma *MinHashTable* para os registar. Para isso, definimos o número de funções de *hash* igual a 100 e o tamanho dos *shingles* igual a 3. O raciocínio para serem calculados foi semelhante ao da alínea anterior.

```
hash=100; %Funções de Hash
shingSize=3;
movies={dic{:,1}};
nMovies=length(movies);

MinHashTable_Movies=inf(nMovies,hash); %tamanho
for i=1:nMovies %tamanho
   movie=movies{i};
   for j=1:length(movie)-shingSize+1
       chave=lower(movie(j:j+shingSize-1));
       h=zeros(1,hash);
       for kk=1:hash
            chave = [chave num2str(kk)];
            h(kk) = DJB31MA( chave, 127);
       end
       MinHashTable_Movies(i,:) = min([MinHashTable_Movies(i,:);h]);
   end
end
```

Fig. 9 - Código do ficheiro suporte para a opção 3 (Cálculo dos MinHash + Shingles)

Calculados os *MinHash* dos filmes, temos que calcular os *minhash* da *string* introduzida pelo utilizador. Isto processa-se de maneira muito semelhante ao que fizemos para calcular os *MinHash*, tanto dos filmes, como dos utilizadores.

Começamos por criar uma matriz de zeros (**M**), com uma linha e duas colunas e uma variável que funciona como um contador (**c**) e após isto, iteramos sobre o número de filmes e calculamos a distância de *Jaccard* através da linha: "***x=sum(MinHashTable_String(1,:)~=MinHashTable_Movies(n2,:))/hash;***" que, mais uma vez, soma o vetor *MinHashTable_String* com a linha *n2* da matriz *MinHashTable_Movie*, apenas onde os valores em ambas as linhas for diferente, dividindo no final pelo número de funções de *hash*. Se esse valor for menor ou igual a 0.99 (condição do enunciado), então a matriz M, vai guardar essa distância de *Jaccard* e o ID do filme, incrementando no fim a variável **c**.

```
str=input("Write a string: ",'s');

hash=100; %Funções de Hash
shingSize=3;
MinHashTable_String=inf(1,hash); %tamanho da minHash

for i=1:length(str)
    for j=1:length(str)-shingSize+1
        chave=lower(str(j:j+shingSize-1));
        h=zeros(1,hash);
            for kk=1:hash
                chave=[chave num2str(kk)];
                h(kk)= DJB31MA( chave, 127);
            end
        MinHashTable_String(1,:)=min([MinHashTable_String(1,:);h]);
    end
end

M=zeros(1,2); %guarda distancia Jaccard e ID filme
c=1;
for n2=1:nMovies
    x=sum(MinHashTable_String(1,:)~=MinHashTable_Movies(n2,:))/hash;
    t(n2)=x;
    if (x<=0.99)
       M(c,:)=[n2 x];
       c=c+1;
    end
end
```

Fig. 10 - Código da Interface para a questão 3

Quando a matriz está finalmente calculada e, se o primeiro valor da primeira coluna for diferente de 0, então significa que há filmes que têm correspondência com a *string* introduzida. Neste caso, são impressos todos os filmes e a respetiva distância de *Jaccard*. Caso contrário, a matriz continua com o seu estado inicial, ou seja, com zero, significando que não foram encontrados nenhuns filmes. Neste caso, é impressa uma mensagem com esse aviso.

```
M=sortrows(M,2,'ascend');

if (M(1,1)~=0)
    for i=1:min(5,length(M))
        fprintf("%10s (%.4f) -> %s\n",'- ' ,M(i,2), string(movies(M(i,1))));
    end
else
    fprintf("\n");
    fprintf('%30s \n','Cannot find any films');
    fprintf("\n");
end
```

Fig. 11 - Código da Interface para a questão 3 (continuação)

Opção 4

Por fim, caso pretendamos terminar o programa, basta fazermos input do número 4 no menu. É impressa a mensagem "*Ending*" e o programa termina.

```
>> interface
Insert User ID (1 to 943): 940
1 - Your Movies
2 - Get Suggestions
3 - Search Title
4 - Exit
Select choice: 4
Ending
```

Fig. 12 - Menu da Interface

Fundamentação da escolha do número de *hash* functions e tamanho dos *shingles*

Experimentamos inicialmente com 200 *hash functions*, no entanto, o código demorava demasiado tempo a correr. De seguida, reduzimos para 150 funções, mas o problema persistiu. Decidimos, assim, utilizar 100 funções de *hash* para as questões 2 e 3, exclusivamente por uma questão de otimização, uma vez que para este valor, o tempo de espera não era tão elevado. Para além disto, 100 funções de *hash* fornece já uma boa precisão para a distância de *Jaccard*.

Quanto ao tamanho dos *shingles*, 3 pareceu-nos um tamanho adequado, porque quanto maior for o tamanho do *shingle*, menor vai ser a probabilidade de haver correspondência com outros *shingles*, isto é, é mais improvável obtermos dois *shingles* iguais (daí excluirmos logo o tamanho igual a 2). Após fazermos algumas experiências com o tamanho dos *shingles* igual a 3 e igual a 4, concluímos que havia mais correspondências com 3, pelo que decidimos utilizar este valor.

Conclusão

Ao longo deste trabalho, a nível teórico, consolidamos os nossos conhecimentos sobre similaridade e *MinHash*, o que são, como funcionam, a sua utilidade e diversidade. A nível prático, melhoramos o nosso conhecimento sobre *Matlab*, aprimorando as nossas habilidades de programação ao implementar novas metodologias de trabalho e pesquisa.