PL04 - Relatório de trabalho prático

Mauro Filho - 103411, Pompeu Costa - 103294

MPEI P3 - 2022/ 2023

UA

PL04 - Relatório de trabalho prático

DETI

UA

Mauro Filho - 103411, Pompeu Costa - 103294

06/01/2023

Conteúdos do relatório

0.1 Introdução

O objetivo deste trabalho é desenvolver uma aplicação em MatLab com funcionalidades de um sistema online de disponibilização de filmes. A aplicação considera um conjunto de utilizadores identificados por um ID (inteiro positivo) e um conjunto de filmes também identificados por um ID (inteiro positivo).

A aplicação permite que um utilizador, após identificar-se pelo seu ID, aced a 4 funcionalidades:

- Opção 1: Lista os títulos dos filmes que o utilizador viu.
- Opção 2: Apresenta sugestões de filmes para o utilizador baseadas em utilizadores com históricos similares.
- Opção 3: Apresenta sugestões de filmes para o utilizador baseadas em filmes avaliados com gêneros similares.
- Opção 4: Permite que o utilizador realize buscas por títulos de filmes.

Para popular as estruturas de dados utilizadas no desenvolvimento deste trabalho utilizamos o ficheiro u.data (release 4/1998) disponível em https://grouplens.org/datasets/movielens/ e o ficheiro films.txt, gerado a partir do anterior.

0.2 Desenvolvimento

0.2.1 Scripts

Foram desenvolvidos dois scripts principais para a execução do trabalho, um para inicializar as estruturas de dados necessárias, e outro para gerir a interação da aplicação com o utilizador.

0.2.1.1 Inicialização de estruturas de dados

A inicialização das estruturas de dados é feita num script dentro do ficheiro "iniEstruturas.m". Neste script vamos ler os conteúdos dos ficheiros de dados disponibilizados (u.data e films.txt), e armazená-los em estruturas que serão posteriormente salvas em um ficheiro "data.mat" para acesso futuro juntamente com um Counting Bloom Filter e listas de assinaturas que darão suporte ao desenvolvimento das funcionalidades da aplicação.

Começamos por carregar os dados de "u.data" e isolar as informações necessárias para criar o Cell Array "C" que guardará os filmes avaliados por cada utilizador. Depois carregamos os dados de "films.txt" em outro Cell Array para guardar o título e géneros de cada filme de acordo com seu ID.

```
udata = load("u.data");
u = udata(1:end, 1:2);
clear udata;
%utilizadores
utiliz = unique(u(:,1));
Nu = length(utiliz);
%lista de filmes
C = cell(Nu, 1);
for n = 1:Nu
    ind = u(:,1) == utiliz(n);
    C{n} = [C{n} u(ind, 2)];
end

dic = readcell("films.txt","Delimiter","\t");
moviesGenre = dic(:,2:end);
moviesName = dic(:,1);
```

Depois de inicializadas as estruturas de dados, criamos o Counting Bloom Filter que dará suporte à contagem de classificações acima de 3 (opção 4 da aplicação)

```
% Loads movies data again
udata2 = load("u.data");
u2 = udata2(1:end, 2:3); clear udata;
bloomFilter = BloomFilter(100000,5);
% Stores every review >= 3 in bloom filter
u2 = u2(u2(:,2) >= 3,1);
dlmwrite('matrix.txt', u2, 'delimiter', '\n', 'precision', 4);
for i = 1:length(u2)
    bloomFilter = bloomFilter.insert(u2(i));
end
```

Por fim, geramos as assinaturas necessárias e guardamos tudo num ficheiro "data.mat"

```
N = 100000;
nhf = 100;
hf = initHashFuncs(N,nhf);

moviesGenreSignaturesMatrix = calculateStringSignaturesMatrix(
    moviesGenre,hf,nhf);
moviesNameSignaturesMatrix = calculateStringSignaturesMatrix(
    moviesName,hf,nhf);
userMoviesSignaturesMatrix = calculateSignaturesMatrix(C,hf,nhf);

```

```
save data.mat userMoviesSignaturesMatrix
    moviesGenreSignaturesMatrix bloomFilter C dic
    moviesNameSignaturesMatrix
```

0.2.1.2 Interações com utilizador

O script que lida com as interações do utilizador, no ficheiro "app.m", pede ao utilizador que se identifique com um ID e depois apresenta um menu com todas as opções de funcionalidades disponibilizadas pela aplicação.

O código de cada um dos "cases" está omitido nesta secção pois será explicado posteriormente.

```
clear all;
id = -1;
while id < 1 || id > 943
    id = input("Insert User ID (1 to 943): ");
end

data = load("data.mat");

option = 1;
while option ~= 5
    disp("1 - Your movies")
    disp("2 - Suggestion of movies based on other users")
    disp("3 - Suggestion of movies based on already evaluated
       movies")
    disp("4 - Movies feedback based on popularity")
    disp("5 - Exit")
    option = input("Select choice: ");

    switch option
        case 1
            ...
        case 2
            ...
        case 3
            ...
        case 4
            ...
    end
end
```

0.2.2 Aplicação

Aqui falaremos sobre a implementação de cada uma das funcionalidades da aplicação.

0.2.2.1 Opção 1

A opção 1 é bastante simples, basta isto buscar a lista de filmes vistos pelo utilizador, guardada em data.C e para cada valor dessa lista apresentar o nome do respetivo filme

```
watchedMovies = data.C{id};
fprintf("\nMovies you watched:\n");
for i=1:length(watchedMovies)
    % data.dic contains all movie's names
    fprintf("%s\n",data.dic{i});
end
```

0.2.2.2 Opção 2

A opção 2 mostra ao utilizador uma lista de recomendações de filmes tendo como base os filmes ainda não vistos pelo mesmo e que já foram vistos por utilizadores semelhantes a ele.

Para isso, primeiro analisamos a similaridade entre o utilizador atual e todos os outros utilizadores através da função "jaccardSimillarity", e a partir dali selecionamos os 2 mais similares, de onde iremos buscar os filmes a analisar.

```
    case 2
            % similarities between the current user and all the
                other users
            % in crescent order
            similarities = jaccardSimillarity(data.
                userMoviesSignaturesMatrix, id);
            % The two most likely users
            likelyUsers = zeros(1,2);
            likelyUsers(1,1) = similarities(1, length(
                similarities));
            likelyUsers(1,2) = similarities(1, length(
                similarities)-1);
            %The movies seen by the 2 most likely users
            movies1 = cell2mat(data.C(likelyUsers(1,1)));
            movies2 = cell2mat(data.C(likelyUsers(1,2)));
            %The movies seem by the current user
            movies3 = cell2mat(data.C(id));
            % Movies seen by at least one of the two likely users
                AND that
```

```
15              % havent been seen by the current user
16              moviesToReccommend = unique(cat(1, movies2, movies1))
                    ;
17              [~,ids1,~] = intersect(moviesToReccommend,movies3);
18              moviesToReccommend(ids1,:) = [];
19              answer = cell(1,length(moviesToReccommend));
20              % Print reccommendations
21              fprintf("\nMovies you might like:\n");
22              for i = 1:length(moviesToReccommend)
23                  fprintf("%s\n",data.dic{moviesToReccommend(i)});
24              end
25              fprintf("\n");
```

A função "JaccardSimilarity" compara as assinaturas geradas pelo script de inicialização do utilizador com as assinaturas de todos os outros utilizadores, e retorna um array ordenado em ordem crescente da similaridade

```
1   function similarities = jaccardSimillarity(signatures, id)
2   % Signatures of current user
3   signatures1 = signatures(:,id);
4   % Number of hash functions
5   nhf = height(signatures1);
6   % array for similarities
7   similarities = zeros(2, length(signatures));
8   % Iterate over the users
9   for j = 1 : length(signatures)
10      if j == id
11          % Doesnt compare the user to itself
12          continue
13      end
14      % Signature of another user
15      signatures2 = signatures(:,j);
16      % Calculate the Jaccard similarity
17      similarity = sum(signatures2(:) == signatures1(:)) / nhf;
18      similarities(1,j) = j;
19      similarities(2,j) = similarity;
20  end
21  % Sort the array in crescent order of the similarity
22  [~, order] = sort(similarities(2,:));
23  similarities = similarities(:,order);
24  end
```

0.2.2.3 Opção 3

A opção 3 implica, para cada filme que o utilizador viu, criar um conjunto de filmes cuja similaridade seja maior que 0.2 ou distância menor que 0.8, criando assim um grupo de conjuntos. Depois é preciso remover filmes ja vistos, pois o filme A pode recomendar o filme B, e caso o utilizador já tenha visto o filme B, não podemos recomendá-lo, De seguida, guardamos os dois filmes que aparecem mais vezes e mostramos os respetivos nomes ao utilizador. Numa situação em que haja vários máximos, selecionamos o primeiro. Vamos imaginar que existem cinco filmes que aparecem 10x cada um e que 10 é o máximo, então guardamos o primeiro ou os dois primeiros.

Isto é feito com o seguinte código:

```
1   % movies the user watched
2   evaluatedMovies = data.C{id};
3   t = [];
4   % for each movie, get simillar movies
5   for i=1:length(evaluatedMovies)
6       simil = jaccardSimillarity(data.
            moviesGenreSignaturesMatrix,evaluatedMovies(i));
7       [~,col,~] = find(simil(2,:) > 0.2);
8       % append the new movies
9       t = [t simil(:,col)];
10  end
11
12  % get the collumns that contain movies the user already
        watched and delete them
13  [~,col,~] = intersect(t(1,:),evaluatedMovies);
14  t(:,col) = [];
15
16  ut = unique(t(1,:));
17
18  % how many times each element appears in t
19  c = histcounts(t(1,:),ut);
20
21  % save the id of the movie(s) that appear most often
22  m = max(c);
23  toRecommend = ut(c == m);
24  if length(toRecommend) == 1
25      c(c == m) = 0;
26      m = max(c);
27      u = ut(c == m);
28      toRecommend = [toRecommend u(1)];
```

```
29      end
30      % prevent more than 2 recommendations
31      toRecommend = toRecommend(1:2);
32      % print the name of the movies to recommend
33      toRecommend = toRecommend(1:2);
34      fprintf("\nMovies you might like:\n")
35      for i=1:length(toRecommend)
36          disp(data.dic{toRecommend(i),1});
37      end
38      fprintf("\n");
```

0.2.2.4 Opção 4

A opção 4 da aplicação permite ao utilizador realizar uma busca por títulos de filmes. Para isso, para cada par de títulos (texto da pesquisa e título de um filme a ser analisado) calculamos a similaridade através da função "minhash" e guardamos num array. Depois selecionamos os 5 resultados mais prováveis e mostramos isso ao utilizador.

```
1       % The search text
2               search = input("Enter your search: ", "s");
3               % Cell array to store similarities
4               similarities = cell(1,length(data.dic));
5               % For each movie title, compare with the search text
                    and
6               % populate the similarities cell array.
7               for i = 1:length(data.dic)
8                   str1 = lower(search);
9                   str2 = lower(data.dic{i});
10                  % Shingle size
11                  shingle_size = 3;
12                  % Number of hash functions
13                  nhf = 10;
14                  % Number of counters in counting bloom filter
15                  nbits = 1000;
16                  % Add the result of the similarity  to the cell
                        array
17                  similarities{i} = minhash(str1, str2,
                        shingle_size, nhf, nbits);
18              end
19              % transform the cell array into an array and sort it
                    in
```

```
20              % descending order
21              [as,idx] = sort(cell2mat(similarities),'descend');
22              % Show the 5 results with a bigger similarity and
                    uses the
23              % bloom filter to count the number of reviews above 3
                     that the
24              % movie has.
25              for i = 1:5
26                  disp(data.dic{idx(i)})
27                  disp(data.bloomFilter.count(idx(i)))
28              end
```

A função "minhash" calcula a similaridade entre duas strings str1 e str2.

Para fazer isso, a função primeiro divide as duas strings em substrings chamadas "shingles" de tamanho "shingle size" usando a função "str2shingle"s. Em seguida, cria dois filtros de Bloom de contagem, que são estruturas de dados que podem armazenar e recuperar dados com eficiência. Em seguida, insere os shingles de cada string de entrada no filtro Bloom correspondente.

Em seguida, a função itera sobre os shingles na primeira string de entrada e verifica se cada shingle também está presente na segunda string usando o método de contagem dos filtros Bloom. Se um shingle estiver presente em ambas as strings de entrada, o contador será incrementado.

Finalmente, a similaridade é calculada como a razão entre o número de shingles comuns e o número total de shingles únicas em ambas as strings de entrada. O resultado é retornado como a saída da função.

```
1   function similarity = minhash(str1, str2, shingle_size, nhf,
        nbits)
2   % Calculate the shingles for the two input strings
3   shingles1 = str2shingles(str1, shingle_size);
4   shingles2 = str2shingles(str2, shingle_size);
5
6   % Create a counting Bloom filter for each string
7   filter1 = BloomFilter(nbits,nhf);
8   filter2 = BloomFilter(nbits,nhf);
9
10  % Insert the shingles into the Bloom filters
11  for i = 1 : length(shingles1)
12      filter1 = filter1.insert(shingles1{i});
13  end
14
15  for i = 1 : length(shingles2)
```

```
16          filter2 = filter2.insert(shingles2{i});
17      end
18
19      % Initialize a counter to track the number of equal shingles
20      counter = 0;
21
22      % Iterate over the shingles in the first string
23      for i = 1 : length(shingles1)
24          shingle = shingles1{i};
25
26          % Check if the shingle is present in the second string
27          multiplicity1 = filter1.count(shingle);
28          multiplicity2 = filter2.count(shingle);
29          if multiplicity1 > 0 && multiplicity2 > 0
30              % If the shingle is present in both strings,
                    increment the counter
31              counter = counter + 1;
32          end
33      end
34
35      % Calculate the Jaccard similarity
36      similarity = counter / (length(shingles1) + length(shingles2)
            - counter);
37  end
```

A função "str2shingles" por sua vez, vai separar cada string em um conjunto de shingles que serão usados pela função "minhash". A função primeiro remove todos os espaços da string de entrada. Em seguida, gera substrings de comprimento "Shingle size" iterando sobre a string de entrada. Para cada iteração, cria uma substring concatenando uma sequência de caracteres da string de entrada, separados por espaços. As substrings resultantes são armazenadas no array shingles e retornadas no final da função.

```
1       function shingles = str2shingles(str, shingle_size)
2       % Removes spaces from string
3       str = str(find(~isspace(str)));
4       l = length(str) - shingle_size + 1;
5       % Generate shingles of shingle_size
6       for i = 1:l
7           t='';
8           for j = i:(i + shingle_size - 2)
9               t = strcat(t,str(j),' ');
10          end
```

```
11          t = strcat(t, str(i + shingle_size - 1));
12          shingles{i} = t;
13      end
14  end
```

E finalmente, a implementação do counting bloom filter segue a mesma abordagem desenvolvida em aula.

A classe tem quatro métodos: "insert", "exists", "count" e o construtor "BloomFilter". O construtor "BloomFilter" inicializa o número de contadores no filtro e o número de funções hash a serem usadas. O método "inser"t adiciona um elemento ao filtro, o método "exist" verifica se um elemento está no filtro e o método "count" determina a multiplicidade (número de ocorrências) de um elemento no filtro usando o algoritmo "minimum selection".

A classe também possui três propriedades: "n", que é o número de contadores no filtro; "k", que é o número de funções hash; e "bits", que é uma matriz de bits que representa o filtro.

```
1       classdef BloomFilter
2       properties
3           n=0
4           k=0
5           bits=[]
6           nElements=0
7       end
8       methods
9
10          % INIT ------------------------------
11          function obj = BloomFilter(nInput, kInput)
12              obj.n = nInput;
13              obj.k = kInput;
14              obj.bits = zeros(1, nInput);
15          end
16
17          % INSERT ----------------------------
18          function obj = insert(obj, toInsert)
19              str = toInsert;
20              for i = 1 : obj.k
21                  str = [str num2str(i)];
22                  hash = string2hash(str);
23                  idx = mod(hash, obj.n) +1;
24                  obj.bits(idx) = obj.bits(idx) + 1;
25              end
26              obj.nElements = obj.nElements + 1;
```

```
27          end
28
29          % EXISTS ------------------------------
30          function returnValue = exists(obj, toCheck)
31              returnValue = True;
32              str = toCheck;
33              for i = 1:obj.k
34                  str = [str num2str(i)];
35                  hash = string2hash(str);
36                  idx = mod(hash, obj.n) +1;
37                  returnValue = (obj.bits(idx) > 0) && returnValue;
38              end
39          end
40
41          % COUNT --------------------------------
42          function minimum = count(obj, element)
43              % Determines the multiplicity of an element using the
                    "minimum selection" algorithm.
44              str = element;
45              minimum = 0;
46              for i = 1 : obj.k
47                  str = [str num2str(i)];
48                  hash = string2hash(str);
49                  idx = mod(hash, obj.n) +1;
50                  value = obj.bits(idx);
51                  if i == 1
52                      minimum = value;
53                  end
54                  if value < minimum
55                      minimum = value;
56                  end
57              end
58          end
59      end
60  end
```

0.2.3 Geração de Assinaturas

Existem duas funções para gerar assinaturas, uma para strings e outra para números.

A função para números (calculateSignaturesMatrix) é igual à usada no guião PL04. Esta pega num conjunto de conjuntos, e para cada elemento,ou seja, para cada conjunto

dentro do conjunto (ex: conjunto utilizadores em que cada utilizador tem um conjunto de filmes vistos; conjunto dentro de outro conjunto), é feito uma hash e guarda-se o mínimo (minhash). Isto é feito para várias hashs.

```
function M = calculateSignaturesMatrix(Conjuntos,hf,nhf)
    nc = length(Conjuntos);
    M = zeros(nhf,nc);

    for nu=1:nc
        C = Conjuntos{nu};
        for nh=1:nhf
            M(nh,nu) = mod(hf.a(nh) * C(1) + hf.b(nh),hf.p);

            for nf=2:length(C)
                htmp = mod(hf.a(nh) * (C(nf)) + hf.b(nh),hf.p);
                if htmp < M(nh,nu)
                    M(nh,nu) = htmp;
                end
            end
        end
    end
end
```

A função para strings (calculateStringSignaturesMatrix) funciona da mesma forma, mas adaptada a strings em vez de números.

```
function minhash = calculateStringSignaturesMatrix(strings,hf,nhf
    )
    minhash = zeros(nhf,length(strings));
    for i=1:length(strings)
        % strings is a cell array of cell arrays
        % ex: {{"Animation","Children's","Comedy"} {"Comedy","
            Animation","Children's","Crime","Thriller","Horror"}}
        st = strings(i,:);
        for nh=1:nhf
            % convert the string into a vector of integers
            s = double(cell2mat(st(1)));
            minhash(nh,i) = mod(hf.a(nh) * sum(s) + hf.b(nh),hf.p
                );
            for nf=2:length(st)
                %ignore missing cells
                if ismissing(st{nf})
                    continue
```

```
                    end
                    s2 = double(cell2mat(st(nf)));
                    htmp = mod(hf.a(nh) * sum(s2) + hf.b(nh),hf.p);
                    if htmp < minhash(nh,i)
                        minhash(nh,i) = htmp;
                    end
                end
            end
        end
    end
```

Nestas funções a hash é feita com a função mod e é utilizado uma hash function gerada previamente. Esta hash function é gerada no ficheiro initHashFuncs:

```
function hf= initHashFuncs(m,nhf)
    %nhf => number of hash functions
    ff=1000;
    pp=ff * max(m+1,76);
    pp=pp + ~mod(pp,2);
    while isprime(pp) == false
        pp = pp + 2;
    end
    hf.p = pp;
    hf.a = randi([1, (pp - 1)],1,nhf);
    hf.b = randi([0, (pp-1)],1,nhf);
end
```