Aplicação de Filmes

METODOS PROBABILÍSTICOS PARA ENGENHARIA INFORMÁTICA

Trabalho realizado por:

Bernardo Miguel Madeira de Figueiredo - 108073

Vitalie Bologa - 107854

universidade de aveiro

2022/2023

Índice

Introdução

No âmbito da disciplina de MPEI (Métodos Probabilísticos Para Engenharia Informática), foi-nos proposto um trabalho prático sobre algoritmos probabilísticos, com o objetivo de desenvolver uma aplicação em MatLab, com funcionalidades de um sistema online de disponibilização de filmes.

O script consiste pedir o ID do filme e através deste, podemos executar as tarefas implementadas no menu inicial:

1 - Listar os nomes de utilizadores que avaliaram o filme atual

2 - Determinar os dois filme mais similares ao filme atual.

3 - Selecionar utilizadores cuja distância de Jaccard seja menor que 0.9

4 - Devolver os três nomes de filmes com os títulos mais similares

5 - A aplicação termina

No decorrer do relatório será apresentado toda a descrição de etapas desenvolvidas na realização deste projeto.

Implementação

No início da implementação começamos por carregar as informações dos ficheiros “u.data”, “film_info.txt” e “users.txt” para as variáveis udata, FilmDic e UserDic, respetivamente.

Opção 1

Na opção 1 é pedido para listar os nomes dos utilizadores que avaliaram o filme atual. Começamos por localizar na da matriz udata, os índices que possuem o valor do ID atual do filme (IDSelect). E as linhas selecionadas anteriormente são guardadas no array users.
Por fim, percorremos todos os utilizadores selecionados, recolhemos o ID e nome, através do array utilizadores e a segunda coluna da matriz UserDic, respetivamente e imprimimos no terminal.

```
case 1
    ind = find(udata(:,2) == IDSelect);
    users = udata(ind);
    for i=1:length(users)
        name = UserDic{users(i),2};
        fprintf("%d - %s\n",users(i), name)
    end
```

Figura 1 - Código visual da opção 1

Opção 2

No programa de processamento é criado a matriz de MinHash de filmes similares em termos de utilizadores que os viram, isto é realizado por aplicar a função de hash nos valores dos ids de utilizadores, utilizando uma técnica de concatenação entre inteiros que consiste em multiplicar o elemento pelo ceil() do logaritmo de base 10 da função de dispersão que está a ser usado no momento mais um e depois adicionar este mesmo número, isto resulta numa concatenação mais rápida do que recorrêssemos ao uso das funções str2num() e num2str() e a concatenação entre strings, que são computacionalmente intensivas passando de 30 segundos de processamento para apenas 1 segundo, isto resulta numa key que será usada na função de hash DJB31MA com uma seed gerada aleatoriamente para cada função de dispersão.

```
function num = stringToNum(String)
%STRINGTONUM Summary of this function goes here
%   Detailed explanation goes here
    arr = double(upper(String));
    num = sum(arr);
end
```

Figura 2 - Função stringToNum(String)

```
%% Opçao 2

FilmList = unique(udata(:,2));
Nu = length(FilmList);
k = 200;
seedMatrix = randi([1 1000],1,k);
FilmMinHash = zeros(Nu, k);

for FilmN=1:Nu
        ind = find(udata(:,2) == FilmList(FilmN));
        for hashFuncN=1:k
            hashArr=zeros(1,length(ind));
            for UserN = 1:length(ind)
                key = udata(ind(UserN),1)*10^(ceil(log10(abs(hashFuncN)))) + hashFuncN;
                hashArr(UserN) = rem(DJB31MA(key, seedMatrix(hashFuncN)), N)+1;
            end
            FilmMinHash(FilmN,hashFuncN) = min(hashArr);
        end
end
save("Opcao2Data", "FilmMinHash")
```

Figura 3 - Cálculo da MinHash de filmes em termos de utilizadores

No programa visual é calculada a distância entre o filme selecionado e o resto dos filmes, de seguida é retirado a lista de utilizadores dos dois filmes mais semelhantes, e verificamos quais destes utilizadores viu o filme escolhido, depois de serem filtrados é apresentado todos os utilizadores que viram os dois filmes mais semelhantes ao selecionado mas não viram o selecionado.

```
case 2
    load("Opcao2Data.mat")
    filmIdMinHash = FilmMinHash(IDSelect,:);
    Distance = zeros(1,length(FilmMinHash(:,1)));
    for i=1:length(Distance)
        isMatch = filmIdMinHash(1,:)==FilmMinHash(i,:);
        Distance(i) = 1-sum(isMatch)/length(isMatch);
    end
    [sortDist,indx] = sort(Distance);
    usersFilm1 = udata(udata(:,2) == indx(2));
    usersFilm2 = udata(udata(:,2) == indx(3));
    usersFilmSel = udata(udata(:,2) == IDSelect);
    userFilms1_2 = unique(union(usersFilm1,usersFilm2));
    users = setdiff(userFilms1_2,usersFilmSel);
    for i=1:length(users)
        name = UserDic{users(i),2};
        fprintf("%d - %s\n",users(i), name)
    end
```

Figura 4 - Código visual da Opção 2

Opção 3

Foi implementada com a ajuda de uma matriz de MinHash onde estão guardados os valores mínimos de várias funções de hash que foram aplicadas aos interesses de cada utilizador. Isto é realizado por tornando cada caracter da string para double, somar cada número que resulta deste processo, com ajuda da função criada, stringToNum(), obtendo um número que corresponde à palavra, é aplicado novamente a técnica de concatenação utilizada na opção 2 no processamento, resultando no número equivalente a palavra a ser concatenado com o número da função de dispersão (eg. Número da palavra = 105, número da função de dispersão atual = 25, key resultante =10525), ao realizar este processo é obtida a key que será usada na função de hash DJB31MA com um seed gerada aleatoriamente.

```
for UserN=1:Nu
    UserData = UserDic(UserN,4:end);
    x = cellfun(@numel,UserData);
    UserData(x==1) = [];
    for hashFuncN=1:k
        hashArr=zeros(1,length(UserData));
        for InteressesN=1:length(UserData)
            key=stringToNum(UserData{InteressesN})*10^(ceil(log10(abs(hashFuncN)))) + hashFuncN;
            hashArr(InteressesN) = rem(DJB31MA(key, seedMatrix(hashFuncN)), N)+1;
        end
        InteressesMinHash(UserN,hashFuncN) = min(hashArr);
    end
end
```

Figura 5 - Cálculo da MinHash de Interesses.

Após ao cálculo da matriz de MinHash dos interesses, é também calculada no processo de dados as distâncias entre utilizadores para poupar tempo no programa visual, a matriz criada será simétrica contendo a similaridade de todos utilizadores, o id do user corresponde à linha/coluna das similaridades dele ao resto dos utilizadores, esta estrutura é utilizada porque embora usa mais memória torna-se mais fácil a manipulação de dados removendo a necessidade de retirar uma coluna e linha para as similaridades do user.

```
UserInterestDistance=zeros(Nu,Nu); % array para guardar distancias
for n1= 1:Nu
    for n2= 1:Nu
        isMatch = InteressesMinHash(n1,:)==InteressesMinHash(n2,:);
        UserInterestDistance(n1,n2) = 1-sum(isMatch)/length(isMatch);
    end
end
save("Opcao3Data", "UserInterestDistance")
```

Figura 6 - Cálculo da aproximação da distância de Jaccard

No programa visual, é criado um dicionário no início que irá armazenar as vezes que utilizadores similares que não viram o filme aparecem, após obter todos os utilizadores que viram o filme selecionado, o programa itera pelos utilizadores com similaridade abaixo de 0.9, e para cada um destes é verificado se não viu o filme, caso isto acontece o id é adicionado ao dicionário com valor correspondente inicializado a 1, se o id já se encontra no dicionário é incrementado o valor correspondente.

Por final ordenamos os dois utilizadores que apareceram mais vezes, e é apresentado os seus ids e nomes.

```
case 3
    load("Opcao3Data.mat")
    ind = find(udata(:,2) == IDSelect);
    users = udata(ind);
    usersNoFilm = dictionary([],[]);
    for i=1:length(users)
        userSimilarity = UserInterestDistance(users(i),:);
        indexOfSim = userSimilarity < 0.9;
        indexOfSim(users(i)) = 0; % remove himself from index;
        simUsers = find(indexOfSim);
        for j=1:length(simUsers)
            watchedFilm = find(udata(udata(:,1) == simUsers(j), 2) == IDSelect);
            if isempty(watchedFilm)
                if isKey( usersNoFilm , simUsers(j))
                    usersNoFilm(simUsers(j)) = usersNoFilm(simUsers(j))+1;
                else
                    usersNoFilm(simUsers(j)) = 1;
                end
            end
        end
    end
    [sorted, indx] = sort(values(usersNoFilm), "descend");
    usersId = keys(usersNoFilm);
    for i=1:2
        name = UserDic{usersId(indx(i)),2};
        fprintf("%d - %s\n", usersId(indx(i)), name)
    end
```

Figura 7 - Código Visual da Opção 3

Opção 4

No código de processamento da opção 4, é criada uma matriz de MinHash com os valores mínimos das funções de hash que foram aplicadas às shingles de cada palavra. Foram escolhidos shingles de tamanho de três caracteres, pois eram um bom meio termo entre quantidade necessária de input e precisão de pesquisa.

A matriz é criada a partir de keys que são formadas pela concatenação da shingle com letras minúsculas e o número de função de dispersão correntemente a ser utilizada, mais uma vez esta key é aplicada à função DJB31MA com valores de seeds aleatoriamente escolhidos para cada função de dispersão.

```
%% Opcao 4

Nu = length(FilmDic);
k = 200;
seedMatrix = randi([1 1000],1,k);
FilmNameMinHash = zeros(Nu,k);
shingleSize = 3;
for FilmNameN=1:Nu
    FilmName = FilmDic{FilmNameN,1};
    for hashFuncN=1:k
        hashArr=zeros(1,strlength(FilmName)-shingleSize+1);
        for ShingleN=1:strlength(FilmName)-shingleSize+1

            key= [lower(char(FilmName(ShingleN:(ShingleN+shingleSize-1)))) num2str(hashFuncN)];
            hashArr(ShingleN) = rem(DJB31MA(key, seedMatrix(hashFuncN)), N)+1;
        end
        FilmNameMinHash(FilmNameN,hashFuncN) = min(hashArr);
    end
end
save("Opcao4Data.mat","FilmNameMinHash","seedMatrix","shingleSize", "k")
```

Figura 8 - Cálculo da MinHash de Nomes de filmes

No processamento também é criado uma matriz de Counting Bloom Filters para armazenar as avaliações todas de cada filme, onde cada linha representa o Counting Bloom Filter do filme com id do número dessa linha, após alguns cálculos chegamos à conclusão que o tamanho da tabela deveria ser 63 e deviam ser aplicadas 9 funções de dispersão, ao mesmo tempo é criado um matriz de seeds que vão ser utilizadas com a função de hash DJB31MA para inserir os elementos no filtro.

```
function [Bloom] = BloomInit(x,y,k)
%BLOOMINIT Summary of this function goes here
%   Detailed explanation goes here
    Bloom = zeros(x,y);
end
```

Figura 9 - Função BloomInit()

```
function [Bloom] = BloomInsert(Bloom, elem, k, seedMatrix, FilmN)
    n = length(Bloom(1,:));
    for i=1:k
        elem = elem*10^(ceil(log10(abs(k)))) + i;
        index = mod(DJB31MA(elem,seedMatrix(k)),n)+1;
        Bloom(FilmN,index) = Bloom(FilmN,index)+1;
    end
end
```

Figura 10 - Função BloomInsert()

```
function [R] = BloomVerify(Bloom, elem, k, seedMatrix, FilmN)
%BLOOMVERIFY Summary of this function goes here
%   Detailed explanation goes here
    R = inf;
    n = length(Bloom(1,:));
    for i=1:k
        elem = elem*10^(ceil(log10(abs(k)))) + i;
        index = mod(DJB31MA(elem,seedMatrix(k)),n)+1;
        if R > Bloom(FilmN,index)
            R = Bloom(FilmN,index);
        end
    end
end
```

Figura 11 - Função BloomVerify()

```
%% bloom
k = 9;
Nu = length(FilmDic);
NSpace= 63;
BloomSeedMatrix = randi([1, 1000], 1, k);
Bloom = BloomInit(length(FilmDic),NSpace,k);

for FilmIdN=1:Nu
    Ratings = find(udata(:,2) == FilmIdN);
    for RatingN=1:length(Ratings)
        Bloom = BloomInsert(Bloom,udata(Ratings(RatingN),3), k, BloomSeedMatrix, FilmIdN);
    end
end

BloomData.Bloom = Bloom;
BloomData.k = k;
BloomData.BloomSeedMatrix = BloomSeedMatrix;
save("BloomData", "BloomData")
```

Figura 12 - Código de processamento do Counting Bloom Filter

No código visual é pedido que o utilizador introduza texto de tamanho mínimo do tamanho das shingles, e é calculado a hash utilizando as mesmas variáveis de seeds que foram usadas no processamento. Após esta operação é calculada a aproximação da distância de Jaccard, onde se retira os 3 filmes mais similares ao que foi introduzido pelo utilizador, de

seguida é verificado o valor de avaliação maiores de que 3 em cada filme é apresentado a informação no terminal.

```
function [minhashArr] = getFilmHash(film_name)
%GETFILMHASH Summary of this function goes here
%   Detailed explanation goes here
    load("Opcao4Data.mat");
    load("GeneralData.mat");
    minhashArr = zeros(1,k);
    for hashFuncN=1:k
        hashArr=zeros(1,strlength(film_name)-shingleSize+1);
        for ShingleN=1:strlength(film_name)-shingleSize+1
            key= [lower(char(film_name(ShingleN:(ShingleN+shingleSize-1)))) num2str(hashFuncN)];
            hashArr(ShingleN) = rem(DJB31MA(key, seedMatrix(hashFuncN)), N)+1;
        end
        minhashArr(1,hashFuncN) = min(hashArr);
    end

end
```

Figura 13 - Função getFilmHash(film_name)

```
case 4
    load("Opcao4Data.mat")
    filmname = lower(char(input("Write film name: ", "s")));
    while (length(filmname) < shingleSize)
        fprintf('Write a minimum of %d characters\n', shingleSize);
        filmname = lower(input("Write a string: ", "s"));
    end
    hashArr = getFilmHash(filmname);
    Distance = zeros(1,length(FilmNameMinHash(:,1)));
    for i=1:length(FilmNameMinHash(:,1))
        isMatch = hashArr(1,:)==FilmNameMinHash(i,:);
        Distance(i) = 1-sum(isMatch)/length(isMatch);
    end
    [distanceClose, closestfilm] = sort(Distance, "ascend");
    filmIds= [closestfilm(1),closestfilm(2),closestfilm(3)];
    load("BloomData.mat")
    for i=1:length(filmIds)
        Rating = 0;
        Rating = Rating + BloomVerify(BloomData.Bloom,3,BloomData.k,BloomData.BloomSeedMatrix, filmIds(i));
        Rating = Rating + BloomVerify(BloomData.Bloom,4,BloomData.k,BloomData.BloomSeedMatrix, filmIds(i));
        Rating = Rating + BloomVerify(BloomData.Bloom,5,BloomData.k,BloomData.BloomSeedMatrix, filmIds(i));
        fprintf("%s : %d Ratings of 3 or more\n", FilmDic{filmIds(i),1}, Rating)
    end
```

Figura 14 - Código Visual da opção 4

Variáveis Escolhidas

No processamento foi sempre utilizado um N com valor de 7919, que é um número primo para calcular o resto das divisão dos códigos de Hash, o número de funções de dispersão foi sempre 200 exceto no Bloom Filter onde são 9, por fim antes de cada processamento é gerada uma matriz de tamanho k de inteiros escolhidos aleatoriamente entre 1 e 1000 que são usados unicamente para a MinHash correspondente, caso seja necessário calcular hash depois do processamento estes arrays são guardados de maneira a que sejam identificáveis a que MinHash correspondem.

Conclusão

No decorrer deste trabalho, implementamos métodos probabilísticos, que consolidou os nossos conhecimentos sobre hash functions, similaridade, filtros de Bloom, etc.

Quanto ao nível prático, melhoramos as nossas capacidade de trabalho enquanto equipa, no desenvolvimento da comunicação e estrutura de pensamento, na produção de algoritmos probabilísticos.

Por fim, acreditamos ter alcançado todos os objetivos dados inicialmente, com um script otimizado.