universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Relatório - MPEI

Desenvolvimento de uma aplicação MATLAB para análise de dados de filmes

Trabalho realizado por:

Ricardo Martins, nº 112876

Rodrigo Jesus, nº 113526

Turma - P4

Ano letivo: 2023/24

Introdução

No âmbito da disciplina de MPEI (Métodos Probabilísticos para Engenharia Informática), foi-nos apresentado um projeto com o objetivo de desenvolver uma aplicação MATLAB com funcionalidades de um sistema de informação de busca de filmes. Esta aplicação irá usar algoritmos probabilísticos e estruturas de dados como filtros de Bloom e métodos minHash para lidar com um conjunto de dados contendo aproximadamente 58.000 filmes

Funcionalidades Chave:

1. Exibir géneros disponíveis.
2. Estimar o número de filmes de um género específico.
3. Estimar o número de filmes de um género específico num determinado ano.
4. Pesquisar títulos de filmes com base na semelhança de strings.
5. Pesquisar filmes com base na semelhança de gênero.
6. Sair da aplicação.

Menu

User Interface:

Implementamos uma interface baseada em menus para interação do usuário, com tratamento de erros incorporado para poder atuar corretamente contra casos de erro como não serem introduzidos valores numéricos para a opção ou o número introduzido estar for limite aceitável.

Opção 1:

São imprimidos todos os nomes dos filmes dentro do nosso array que contém apenas os géneros de filmes sem repetição.

Opção 2:

Obtemos o input limpo do utilizador e verificamos se o género por ele introduzido pertence ou não ao conjunto de géneros válidos, se não pertencer imprime uma mensagem de erro. Caso pertença o número de filmes pertencentes a esse género vai ser estimado através do uso de um filtro de Bloom de contagem específico para géneros apenas.

Opção 3:

Obtemos o input limpo do utilizador e verificamos se o género e ano por ele introduzido pertence ou não ao conjunto de géneros, anos válidos, se não pertencer a qualquer um dos conjuntos será imprimida uma mensagem de erro especifica ao caso. Caso os valores introduzidos sejam válidos, o número de filmes pertencentes a esse género e no ano especificado, irá ser estimado através do uso de um filtro de Bloom de contagem específico para géneros e anos.

Opção 4:

Esta secção do código utiliza a similaridade de Jaccard para encontrar títulos de filmes semelhantes com base numa string fornecida pelo utilizador. Utiliza a técnica MinHash para calcular de forma eficiente a similaridade entre títulos de filmes.

Opção 5

Obtemos o input limpo do utilizador e verificamos se o género por ele introduzido pertence ou não ao conjunto de géneros válidos, se não pertencer imprime uma mensagem de erro.

```
movies = readcell('movies.csv', 'Delimiter', ',');
titles = movies(:,1);
numTitles = length(titles);
years = movies(:,2);
uniqueYears = cellfun(@safeStr2Num, movies(:, 2), 'UniformOutput', true);
uniqueYears = unique(uniqueYears);
numYears = height(uniqueYears);

genres = getGenres(movies);
uniqueGenres = unique(genres);
uniqueGenres = setdiff(uniqueGenres, {'(no genres listed)'});
numGenres = length(uniqueGenres);
```

```
% 2 and 3
filterSize = 1000;
numHashFuncGenre = round(filterSize * log(2) / numGenres);   % valor otimo de k teorico
numHashFuncGenreYear = round(filterSize * log(2) / (numGenres+numYears));   % valor otimo de k teorico
fprintf("Initializing Bloom filters....")
genreFilter = inicializarFiltro(filterSize);
genreYearFilter = inicializarFiltro(filterSize);
fprintf("Done\n")

fprintf("Populating bloom filters....")
% Bloom filter for genres
for i = 1:length(genres)
    genreFilter = adicionarElemento(genreFilter, genres{i}, numHashFuncGenre);
end
% Populate Bloom filter for genre-year combination
for i = 1:height(movies)
    if ~isempty(genres{i}) && ~isempty(years{i})
        genreYearFilter = adicionarElementoAno(genreYearFilter, genres{i}, years{i}, numHashFuncGenreYear);
    end
end
fprintf("Done\n")
```

```
function genres = getGenres(movies)
    genres = {};
    k = 1;

    for i= 1:height(movies)
        for j= 3:12
            if ~anymissing(movies{i,j}) && ~strcmp(movies{i,j},'unknown')
                genres{k} = movies{i,j};
                k = k+1;
            end
        end
    end

end
```

Propósito: Esta função extrai os géneros de filmes de uma matriz de células **movies**, que contém informações sobre filmes.

Como Funciona:

1. **Inicialização**: A função começa por criar um array de células vazio chamado **genres**, que irá armazenar os géneros extraídos.
2. **Loop Duplo**:
    - O loop externo (**for i= 1:height(movies)**) itera sobre cada filme na matriz **movies**.
    - O loop interno (**for j= 3:12**) percorre as colunas da 3ª à 12ª de cada filme, que são assumidas conter os géneros.
3. **Extração de Generos**:
    - Dentro dos loops, a função verifica se a célula atual (**movies{i,j}**) não está vazia e não é igual a **'unknown'** usando a condição **~anymissing(movies{i,j}) && ~strcmp(movies{i,j},'unknown')**.
    - Se estas condições forem verdadeiras, o género é adicionado ao array **genres**, e o índice **k** é incrementado.

Utilização: Esta função é útil para criar uma lista de todos os géneros disponíveis nos dados dos filmes, removendo qualquer género desconhecido ou não listado.

```
function filtro = inicializarFiltro(n)
    filtro = zeros(n,1); % Iniciar o filtro como um array de zeros
end
```

Primeiro temos a função *inicializarFiltro* que inicializa um *array* de zeros com o tamanho especificado pelo parâmetro de entrada **n**. Isto criará o filtro de *Bloom* com **n** posições, que é que é uma estrutura de dados usada para testar rapidamente se um elemento está presente num conjunto.

```
function filter = adicionarElemento(filter, key, numHashFunc)
    for i= 1:numHashFunc
        % coloca a chave igual à concatenação da chave com o indice
        key = [key num2str(i)];

        % usa uma função hash para codificar a chave concatenada
        code = mod(string2hash(key), length(filter)) + 1;

        % Incrementa a contagem no índice 'code' do filtro Bloom.
        filter(code) = filter(code)+1;
    end
end
```

Depois temos a função *adicionarElemento*. Esta função é usada para adicionar um elemento ao filtro de *Bloom* e este é implementado como um *array* de inteiros. Quando um elemento é adicionado ao filtro, o valor das posições correspondentes é incrementado em um.

Através de um *for* loop, este itera pelo número de funções de dispersão especificado pelo parâmetro numHashFunc. Em cada iteração, o *for loop* gera um novo valor de *hash* para a chave e incrementa-os nas posições do filtro correspondentes.

O parâmetro *length(filter)* é usado para garantir que o valor de *hash* gerado pelas funções de dispersão não excede o tamanho do filtro.

```
function resultado = pertenceConjunto(filter,key,numHashFunc)
    res_lst = zeros(numHashFunc,1); % inicializa um vetor para guardar os resultados

    for i = 1:numHashFunc
        % coloca a chave igual à concatenação da chave com o indice
        key = [key num2str(i)];

        % usa uma função hash para codificar a chave concatenada
        code = mod(string2hash(key),length(filter))+1;

        res_lst(i) = filter(code); % Store the result in its corresponding index
    end

    % verifca se a soma de todos os resultados é igual ao numero de funções hash.
    resultado = min(res_lst);
end
```

A função *pertenceConjunto* é usada para verificar se um elemento pertence a um filtro bloom.

Através de um *for loop* que itera pelo número de funções de dispersão especificado pelo parâmetro numHashFunc, cada *loop* gera um novo valor de hash para a chave e incrementando as posições correspondentes no filtro. Os valores são armazenados num vetor *res*.

Os valores de hash são então usados para acessar às posições no filtro de Bloom.

Por fim, a função retorna o valor mínimo de *res*. Se todos os valores de *res* são diferentes de 0, então há grande probabilidade de haver filmes daquele género e a quantidade estimada é igual ao mínimo desses valores. Caso contrário, não há nenhum filme desse género.

Foram também criadas duas funções alternativas AdicionarElementoAno e pertenceConjuntoAno que são em tudo semelhantes às duas anteriores exceto que em vez de aceitarem como argumento de entrada key que seria o género, agora aceitam género e ano como argumentos de entrada e por isso criam uma chave resultante da concatenação do género, com o ano, com o índice.

```
numHash = 20;
shingleSize = 2;
matrizMinHashTitles = minHashTitles(titles,numHash,shingleSize);
distancesTitles = getDistancesByTitles(numTitles,matrizMinHashTitles,numHash);
```

Função minHashTitles

Função:

- Calcula a matriz MinHash para todos os títulos de filmes.

Como Funciona:

1. **Inicialização**: Cria uma matriz **matrizMinHashTitles** para armazenar os valores MinHash de cada título.
2. **Processamento de Cada Título**:
    - Para cada título, cria shingles (subconjuntos de caracteres) e aplica várias funções hash.
    - Armazena o menor valor hash para cada função na matriz MinHash.

```
function matrizMinHashTitles = minHashTitles(titles,numHash,shingleSize)
    numTitles = length(titles);
    matrizMinHashTitles = inf(numTitles, numHash);

    b = waitbar(0,'Calculating minHashTitles()...');
    for i= 1 : numTitles
        movie = titles{i};
        waitbar(i/numTitles,b);
        for j = 1 : (length(movie) - shingleSize + 1)
            h = zeros(1, numHash);
            shingle = lower(char(movie(j:(j+shingleSize-1))));
            for nHash = 1 : numHash
                shingle = [shingle num2str(nHash)];
                h(nHash) = DJB31MA(shingle, 127);
            end
            matrizMinHashTitles(i, :) = min([matrizMinHashTitles(i, :); h]);
        end
    end
    delete(b);
end
```

Função getDistancesByTitles

Função:

- Calcula uma matriz de distâncias entre todos os pares de títulos de filmes com base nas suas assinaturas MinHash.

Como Funciona:

1. **Cálculo de Distância**: Para cada par de títulos, calcula a proporção de hash matches entre as suas assinaturas MinHash, o que é uma aproximação da sua similaridade de Jaccard.

```
function distances = getDistancesByTitles(numTitles,matriz,numHash)
    distances = zeros(numTitles,numTitles);
    for n1= 1:numTitles
        for n2= n1+1:numTitles
            distances(n1,n2) = sum(matriz(n1,:)==matriz(n2,:))/numHash;
        end
    end
end
```

Função searchTitle

Encontra e apresenta os títulos de filmes mais semelhantes à string de pesquisa, com base na similaridade de Jaccard. Como funciona:

1. **Calcula MinHash da Pesquisa**: Gera a assinatura MinHash da string de pesquisa.
2. **Comparação com Títulos Existentes**: Usa a função **filterSimilar** para encontrar títulos semelhantes na base de dados.
3. **Exibição de Resultados**: Apresenta os títulos encontrados e as suas pontuações de similaridade.

```
function searchTitle(search, matrizMinHashTitles, numHash, titles, shingleSize)
    minHashSearch = inf(1, numHash);
    for j = 1 : (length(search) - shingleSize + 1)
        shingle = char(search(j:(j+shingleSize-1)));
        h = zeros(1, numHash);
        for i = 1 : numHash
            shingle = [shingle num2str(i)];
            h(i) = DJB31MA(shingle, 127);
        end
        minHashSearch(1, :) = min([minHashSearch(1, :); h]);
    end

    threshold = 0.99;
    [similarTitles,distancesTitles,k] = filterSimilar(threshold,titles,matrizMinHashTitles,minHashSearch,numHash);

    if (k == 0)
        disp('No results found');
    elseif (k > 5)
        k = 5;
    end

    distances = cell2mat(distancesTitles);
    [distances, index] = sort(distances);

    for h = 1 : k
        fprintf('%s - Similaridade: %.3f\n', similarTitles{index(h)}, 1-distances(h));
    end
end
```

Função filterSimilar

Função:

- Filtra e encontra títulos semelhantes com base num limiar de similaridade.

Como Funciona:

1. **Comparação de MinHash**: Compara a assinatura MinHash da string de pesquisa com as dos títulos existentes.
2. **Seleção Baseada em Similaridade**: Seleciona títulos cuja distância de Jaccard esteja abaixo de um limiar especificado.

```
function [similarTitles,distancesTitles,k] = filterSimilar(threshold,titles,matrizMinHashTitles,minHash_search,numHash)
    similarTitles = {};
    distancesTitles = {};
    numTitles = length(titles);
    k=0;
    for n = 1 : numTitles
        distancia = 1 - (sum(minHash_search(1, :) == matrizMinHashTitles(n,:)) / numHash);
        if (distancia < threshold)
            k = k+1;
            similarTitles{k} = titles{n};
            distancesTitles{k} = distancia;
        end
    end
end
```

- Search -> Este é o termo de pesquisa, que deve ser um array de caracteres;
- matrizMinHashTitles -> Esta é uma matriz de valores minHash para um conjunto de títulos. Cada linha representa um título e cada coluna representa uma função de dispersão;
- numHash -> Este é o número de funções de dispersão usadas para calcular os valores minHash;
- Titles -> Este é um array de títulos, onde cada célula representa um título;
- shingleSize -> Este é o tamanho do shingle usado para calcular os valores minHash para o termo de pesquisa.

Explicação dos Valores Implementados

Na implementação dos filtros de Bloom a serem usados nas opções 2 e 3, decidimos inicializar filtros de tamanho 1000 pois após algumas experiências concluímos que esse seria um valor aceitável a ser usado, pois produz números já próximos os suficientes dos reais e não consome tanto tempo como se utilizássemos valores mais elevados. Tendo verificado se os valores estavam corretos ou não ao localizar os nomes e anos dentro do ficheiro movies.csv.

Quanto ao número de funções de dispersão para os filtro de Bloom decidimos utilizar a fórmula usada para obter o número ótimo de k funções de dispersão:

round(filterSize * log(2) / numGenres) para o filtro dos géneros apenas.

round(filterSize * log(2) / (numGenres+numYears)) para o filtro dos géneros e anos.

Para as funções de minHash decidimos usar um número de hash functions igual a 20 pois já é capaz de produzir valores fidedignos e usar números superiores a este demoravam demasiado tempo o que impossibilitava o seu uso.

Quanto ao tamanho dos shingles, 2 pareceu-nos um tamanho adequado, porque após fazermos algumas experiências com o tamanho dos *shingles* igual a 3 e igual a 2, concluímos que havia mais correspondências com 2, pelo que decidimos utilizar este valor.

Isto foi possível de observar em casos onde introduzíamos Iron-Man ao usarmos 2 shingles obtemos um valor de similaridade ao titulo Iron Man de 0.7 enquanto que se usarmos shingles de tamanho 3 já só tínhamos uma similaridade de 0.35, logo o shingle de tamanho 2 estaria mais correto pois apenas têm um caracter diferente.

O mesmo acontecia para outros títulos que eram bastante semelhantes com a string que introduzíamos e quais a sua similaridade era reproduzida com mais precisão para shingles de tamanho 2.

Conclusão

A realização deste projeto ajuda-nos a consolidar os nossos conhecimentos em tópicos essenciais como as hash functions, os filtros de Bloom, similaridade e distância de Jaccard, bem como o método MinHash.

Ao nível do desenvolvimento e qualidade do projeto acreditamos ter sido um sucesso pois fomos capazes de criar com sucesso uma interface amigável para permitir ao utilizador interagir de diferentes maneiras com um grande conjunto de dados de filmes, através da implementação de filtros de Bloom de contagem e do uso de métodos MinHash para fornecerem uma manipulação e consulta de dados eficientes.