Turma P7 – Gonçalo Silva (98008) e Cláudia Ferreira (98678)

Universidade de Aveiro – Métodos Probabilísticos para Engenharia Informática

Janeiro 2021



# Introdução

O trabalho que vamos realizar no âmbito da cadeira de Métodos Probabilísticos para Engenharia Informática tem como objetivo a criação de uma aplicação. Para a realização desta aplicação foi-nos proposto que implementássemos algumas funcionalidades de um sistema de disponibilização de filmes online. A partir da matéria abordada nas aulas teóricas e consolidada nas aulas práticas devíamos ser capazes de desenvolver algoritmos de sugestão e pesquisa de filmes.

Atualmente, este tipo de algoritmos, claro que mais complexos que os algoritmos desenvolvidos para este trabalho prático, são uma das formas mais rentáveis de utilizar a internet. Neste caso, são feitas sugestões de filmes a um utilizador baseadas nos filmes que este utilizador já viu. No entanto, se pensarmos em grande escala, temos exemplos como o YouTube que, com base nos filmes que costumamos ver, nos apresenta sugestões de vídeos dentro do mesmo tema e publicidade que à partida será do nosso interesse.

No ramo da publicidade estes algoritmos são essenciais, de nada serve anunciar um produto se aquele anúncio não for visto por ninguém ou se for visto por pessoas que não têm interesse naquele tipo de produtos. Se conseguirmos falar apenas com os interessados num produto a taxa de sucesso será bastante maior e a publicidade será mais eficaz.

Outra razão para que algoritmos probabilísticos de similaridade sejam tão importantes é a quantidade astronómica de informação que circula na internet nos dias de hoje. Por exemplo, se entrarmos no YouTube apenas para ver um vídeo para descomprimir, mas sem grande ideia do que queremos ver, é com bom grado que recebemos as suas sugestões de vídeos que nos deverão interessar. Se tivéssemos de andar à procura de um vídeo que seja do nosso interesse sem que existissem sugestões o processo iria ser muito mais demorado e plataformas deste género não teriam tanto sucesso, porque os utilizadores se fartariam rapidamente.

Para resolver os desafios propostos usamos o MATLAB para implementar funções assentes principalmente em *MinHashes* e distância de Jaccard. A utilização de minHash permite reduzir o tempo de computação para tempos aceitáveis. Se quiséssemos usar shingles para procurar semelhanças entre um ficheiro de texto e uma pasta com 1 milhão de documentos iriamos demorar alguns anos a percorrer os ficheiros todos para descobrir a distância de Jaccard entre eles. Através do uso de minHash o tempo necessário reduz substancialmente, passando a ser uma comparação possível de ser feita porque iria demorar no máximo umas horas e não anos.

Neste relatório iremos primeiramente apresentar os dois scripts desenvolvidos. De seguida será mostrado os métodos realizados para o restante funcionamento do programa e uma breve exemplificação dos resultados que foram obtidos. Por fim, apresentaremos as conclusões que conseguimos retirar deste projeto.

# Scripts

**Script 1** – readData\_script1.m

O primeiro script desenvolvido tem como objetivo realizar a leitura dos ficheiros de entrada, mais concretamente dos ficheiros “u.data” e “u\_item”. O ficheiro “u.data” apresenta todos os dados dos filmes e o ficheiro “u\_item” apresenta todos os filmes e as correspondentes características relativamente ao género de cada um. Além disso, ainda foi necessário guardar em ficheiro todas as estruturas de dados associadas aos utilizadores e aos filmes, incluindo a matriz com os vetores MinHash de cada utilizador e a matriz com os vetores MinHash de cada título, tal como pedido no enunciado do trabalho.

Para isso foi desenvolvido o seguinte script:

Uma imagem com texto

Descrição gerada automaticamente

Figura - Script 1

Uma imagem com texto

Descrição gerada automaticamente

Figura - Script 1 (continuação)

Observando ambas as figuras, pode-se ver que foram guardadas todas as informações que serão necessárias ao longo do trabalho (linha 39). O código relativamente às matrizes com os vetores minHash serão explicados mais à frente.

**Script 2** – app\_scrip2.m

O segundo script começa por ler do disco todas as estruturas previamente guardadas pelo primeiro script e depois implementa todas as interações com o utilizador.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Script 2

Uma imagem com texto

Descrição gerada automaticamente

Figura - Script 2 (continuação)

Na figura 3 e 4 pode-se então ver que a aplicação será composta por quatro opções, tal que a última apenas tem a função de terminar o programa se o cliente desejar (linha 51 e 52).

Nas restantes opções será possível verificar, de acordo com o número de identificação de cada utilizador, os filmes que lhe pertencem, as sugestões que poderão ser dadas e ainda a procura de títulos. Todas estas opções serão melhor explicadas mais à frente.

É de notar que também foi feita uma correta implementação de todas as interações com o utilizador e a validação de todos os dados inseridos pelo cliente.

# Métodos

Nesta secção serão mostradas todas as funções desenvolvidas.

Primeiramente iremos apresentar a função getUserMovies que permite obter um conjunto (cell array) com todos os utilizadores e correspondentemente a quantidade de filmes vistos por cada um. Esta função será chamada no script um (figura 1 – linha 5) de forma a guardar o seu resultado para poder ser usado ao chamar a função listMovies para a resolução da opção 1 do menu da aplicação.

A *opção 1* tem como objetivo listar todos os títulos dos filmes que o utilizador atual viu. A funções getUserMovies e listMovies são as seguintes:

* **getUserMovies.m**

Uma imagem com texto

Descrição gerada automaticamente

Figura - Função getUserMovies

A função getUserMovies percorre todos os utilizadores e encontra através da função find do Matlab todos os filmes correspondentes a cada pessoa e retorna desse modo um cell array que representa a lista de filmes para cada utilizador (como referido no comentário da linha 7 do código da figura 5).

* **listMovies.m**

**Uma imagem com texto

Descrição gerada automaticamente**

Figura - Função listMovies

Esta função tem como parâmetros o conjunto de utilizadores e os seus respetivos filmes vistos (valor obtido da função getUserMovies), os filmes obtidos a partir do ficheiro “u\_item” lido no script 1 e ainda o número de identificação do utilizador atual. Serão percorridos todos os números de identificação dos filmes (id´s) do ID atual (linha 8 a 12) de forma a encontrar os nomes correspondentes a esses filmes (linha 9) e listá-los, um por linha. Ainda neste método determinámos a quantidade de filmes vistos pelo ID atual.

Na *opção 2* é pedido para o cliente selecionar um dos 19 géneros que são apresentados e que caracterizam os filmes. A aplicação, de seguida, tem a função de determinar qual de todos os outros utilizadores é mais similar ao ID atual relativamente ao conjunto de filmes vistos por cada um. Deste modo, deverão ser obtidos e apresentados todos os títulos dos filmes do género que foi selecionado inicialmente e que o utilizador atual (ID) ainda não viu. Assim, para a realização desta opção foi desenvolvida a função suggestMovies.

* **suggestMovies.m**

Uma imagem com texto

Descrição gerada automaticamente

Figura - Função suggestMovies

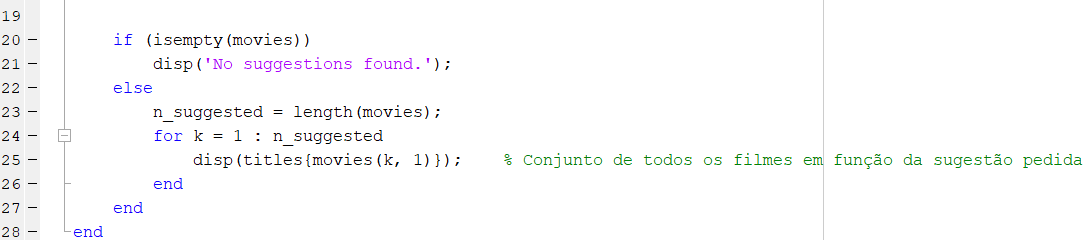


Figura - Função suggestMovies (continuação)

A função acima (figura 7 e 8) apresenta como parâmetros o número representativo da sugestão do género pretendido (genre\_id), a matriz com os valores minHash de cada utilizador (minHash), o número de funções de dispersão percorridas (n\_hash), as duas primeiras colunas do ficheiro “u.data” (u) em que a primeira apresenta o ID de cada utilizador e a segunda o ID do filme que foi avaliado pelo utilizador da primeira coluna. Ainda apresenta as colunas dois a vinte do ficheiro “u\_item” dos géneros disponíveis (u\_genres), o ID do utilizador atual e por fim todos os títulos dos filmes do ficheiro “u\_item” (titles).

É de notar que o parâmetro minHash, n\_hash, u, u\_genres e titles são todos obtidos no script para ler os dados (script 1) e carregados no script 2 para serem usados na chamada da função suggestMovies da opção 2 como já foi referido.

Inicialmente foi então necessário obter o utilizador mais similar ao utilizador atual. Para isso foi utilizada a função getSimilarUsers (figura 9) de forma a devolver uma matriz com a distância de Jaccard entre cada utilizador e o utilizador atual escolhido. Obtendo essa matriz e encontrando (através da função find) um valor diferente de zero comparando cada um desses valores da matriz com o mínimo de um desses valores encontra-se o utilizador pretendido (linha 3 e 4 – figura 7). De seguida determinam-se os números de identificação dos filmes que o ID atual ainda não viu. Através da função setdiff do Matlab é possível obter os valores desses id´s dos filmes comparando o conjunto dos filmes do utilizador mais similar ao do utilizador atual.

Desta forma, percorrendo todos os filmes não vistos pelo utilizador atual e verificando se o filme é do género selecionado, então obtém-se os filmes pretendidos. No caso de não haver nenhum título nestas condições a aplicação indica que não tem nenhuma sugestão.

Como referido anteriormente foi utilizada a função getSimilarUsers que será explicada de seguida:

* **getSimilarUsers.m**

A função getSimilarUsers retorna uma matriz tal que a primeira coluna representa o número de identificação (ID) do utilizador atual, a segunda cada um dos outros utilizadores e a terceira a distância de Jaccard entre cada um dos utilizadores.

Deste modo, esta função irá permitir determinar os utilizadores mais similares entre o utilizador fornecido, tal será possível através dos cinco parâmetros recebidos. Assim, recebe os valores das duas primeiras colunas do ficheiro “u.data” (u) que já foram explicados para o caso da função suggestMovies, a matriz devolvida da função minHash, k funções de dispersão, o valor do limiar de decisão (threshold) e o ID atual. O limiar de decisão será usado para podermos obter todos os pares de valores com distância igual ou inferior a esse determinado limiar. O foi referido pode ser observado no código da figura 9.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Função getSimilarUsers

* **minHash.m**

A função minHash calcula as Hash do conjunto de filmes vistos por cada um dos utilizadores usando hash functions. Para isso utiliza n\_hash functions e após calcular esses valores vê qual dos conjuntos é menor, o atual ou o anterior. MinHash é isso mesmo, escolher o menor valor de hash para um conjunto.

Recebe como argumentos o conjunto de todos os utilizadores e as correspondentes quantidades de filmes vistos por eles (Set) e ainda o número de funções de dispersão percorridas (n\_hash).

A quantidade de funções de dispersão percorridas será explicada na conclusão depois de serem feitas as simulações necessárias para chegarmos a esse valor final.

A função de dispersão utilizada é a DJB31MA que foi dada já desenvolvida.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Função minHash em relação aos filmes dos utilizadores

Na *opção 3* o objetivo baseia-se no cliente introduzir uma palavra e através dela procurar os títulos dos filmes que lhe sejam mais similares. Em conjuntos com esses títulos deve-se obter uma estimativa da distância de Jaccard entre essa string e cada título obtido. A lista determinada deverá ter no máximo cinco filmes e têm de estar ordenados por ordem crescente de distância de Jaccard cuja distância seja menor ou igual a 0.99.

Para a obtenção destes resultados foi desenvolvida uma função (searchTitle) e um método minHash adequado à similaridade entre vetores e caracteres.

A função searchTitles recebe como parâmetros a string introduzida no script dois (search), a matriz obtida com os vetores MinHash de cada título (minHash\_titles), o número de hash functions (n\_hash\_titles), o conjunto de títulos e o tamanho dos shingles (n\_shingles). O segundo e terceiro parâmetros foram obtidos no script um e carregados para o script dois, tal que foi necessário desenvolver nesse script um método minHash para o caso dos títulos com shingles. O método desenvolvido apresenta-se na figura 2 entre as linhas 14 e 35.

Para a realização desta função foi necessário criar um método minHash com o intuito de obter os MinHashes dos títulos para descobrir os mais similares com a palavra que foi introduzida. Para isso foi ainda preciso obter um tamanho adequado de shingles. Tal que esse valor em conjunto com o número de funções de dispersão usadas para esta opção na implementação deste método minHash será explicado na conclusão.

Depois de serem obtidos os MinHashes foram então determinados todos os títulos mais semelhantes em relação à palavra que foi introduzida e em conjunto a distância estimada entre cada título encontrada e a respetiva palavra usando um limiar de decisão 0.99.

Os títulos encontrados e as respetivas distâncias estimadas serão ordenadas de forma crescente por distância e listados um por linha até um máximo de 5. Caso não existam títulos semelhantes (distância de Jaccard menor que 0.99) será dito que não foram encontrados resultados.

Nas figuras 11, 12 e 13 é possível ver o código da função searchTitle.

* **searchTitle.m**

Uma imagem com texto

Descrição gerada automaticamente

Figura - Função searchTitle

Uma imagem com texto

Descrição gerada automaticamente

Figura - Função searchTitle (continuação1)

Uma imagem com texto

Descrição gerada automaticamente

Figura - Função searchTitle (continuação2)

# Resultados obtidos

Nesta secção iremos simular através de um número de identificação do utilizador escolhido as diversas funcionalidades deste sistema de disponibilização de filmes online. É de relembrar que a opção 4 é apenas uma funcionalidade de terminar o programa.

Escolhendo o ID número 4 pode-se obter os seguintes resultados:

* **Opção 1**

**Uma imagem com texto

Descrição gerada automaticamente**

Figura - Nome e número dos filmes vistos pelo utilizador número 4

* **Opção 2**

Iremos mostrar os filmes obtidos relativamente à sugestão “Action” (sugestão 2) e “Documentary” (sugestão 8):

Uma imagem com texto

Descrição gerada automaticamente

Figura 15 - Filmes obtidos para a sugestão 2 e 8

Como se pode observar nos resultados obtidos, caso para a sugestão pedida não seja encontrada nenhum valor, a aplicação indica que não tem nenhuma sugestão.

* **Opção 3**

Sendo a palavra introduzida, por exemplo, “air” (sequência de 3 caracteres) ou “star” (sequência de 4 caracteres) obtém-se o seguinte resultado:

**Uma imagem com texto

Descrição gerada automaticamente**

Figura 16 - Resultados obtidos para a pesquisa da palavra "air" e "star"

De seguida iremos mostrar o caso em que se introduzirmos uma palavra com menos de 2 caracteres será apresentado um erro e o caso em que se não for encontrado nenhum filme similar à palavra introduzida, a aplicação indicará que não encontrou nenhum título.

Uma imagem com texto

Descrição gerada automaticamente

Figura 17 - Caso de erro e da não obtenção de um resultado de similaridade

# Conclusões

Ao longo de todas as simulações que fomos fazendo, que tiveram o objetivo de observar os resultados desta aplicação, tivemos de ter em conta determinados números que fariam os valores obtidos mudar ligeiramente ou mesmo de forma significativa. Esses números dizem respeito à quantidade de funções de dispersão usadas para o método desenvolvido na minHash da opção 2, o número de funções de dispersão para o método minHash implementado na opção 3 e ainda tivemos de decidir qual o melhor tamanho (entre 2 e 5 caracteres) para os shingles usados no método minHash da opção número três.

Relativamente à segunda opção, testando para valores até 200 funções de dispersão, chegou-se à conclusão de que quanto maior a quantidade de hash functions percorridas, menor será a quantidade de filmes obtidos para qualquer sugestão referida. Isto acontece devido ao facto de ao percorrer um conjunto maior de funções de dispersão, maior será a precisão de valores mais similares obtidos relativamente aos filmes disponíveis e à sugestão sugerida. Desta forma, escolhemos percorrer apenas 100 funções de dispersão, para que o número de filmes obtidos relativamente à sugestão referida seja mais preciso e de forma a que na simulação o tempo de execução seja menor, pois o aumento do número de hash functions afeta negativamente o tempo de execução.

Em relação à simulação para a opção 3 tem de se ter em atenção que se escolhermos genericamente shingles de tamanho n, a palavra introduzida tem de ser constituída por pelo menos n caracteres, caso contrário será gerado um erro, pois para transformar os títulos em hash utilizamos sequências de n caracteres e se comparássemos esses valores com minHash obtidos através de shingles menores não iríamos ter valores corretos.

Sendo assim, é necessário escolher de forma adequada o tamanho dos shingles para que ao introduzir uma determinada string, em qualquer simulação que se esteja a realizar, seja possível identificar sequências idênticas nos títulos disponíveis. Ou seja, o tamanho dos shingles tem de ser um valor que nos permita ter uma probabilidade alta, mas fiável, de encontrar sequências de carateres idênticas à introduzida. Realizando várias simulações para diferentes valores entre 2 e 5 sequências de caracteres, concluiu-se que o melhor valor seria atribuir o valor 3 às shingles.

Mais concretamente, se pensarmos melhor nestes números, consegue-se perceber que para um shingle de tamanho 2 a probabilidade de encontrarmos uma sequência semelhante num dos títulos é muito maior do que se escolhermos sequências de 3, 4 ou 5 caracteres. Por outro lado, se pensarmos em escolher um tamanho de shingle 4 ou 5, a probabilidade de termos palavras com aquela sequência de 4 ou 5 carateres relativamente ao valor 2 é realmente pequena. No entanto, a probabilidade desse conjunto ser tão pequeno diminui em relação a shingles de tamanho 3, pois é necessário introduzir palavras com pelo menos 3 caracteres o que garante que o conjunto de palavras com uma sequência semelhante é grande, mas não suficientemente grande para conter títulos que em nada se assemelham à string introduzida como acontecia no caso dos shingles terem tamanho 2 e por estas razões foi escolhido o tamanho dos shingles igual a 3.

Escolhido o tamanho do shingle, foi então possível, para as diferentes simulações efetuadas, concluir acerca do valor do número de funções de dispersão a percorrer para o método minHash implementado para a realização da opção 3. Assim, simulando para diferentes valores percebeu-se que para cada atribuição diferente do número de hash functions obtém diferentes resultados. Tal que, ao aumentar o número de funções de dispersão é aumentada também a precisão da similaridade entre a palavra introduzida e o título do filme obtido. Consequentemente, é diminuída a distância entre cada palavra e filme aumentando de igual forma a similaridade. Portanto, testando para diferentes valores e aumentando esse valor, os resultados obtidos estarão sempre a ser alterados, mas sempre com uma melhor precisão dos resultados mais similares. Desta forma, escolhemos o valor 100 como o número de funções de dispersão a percorrer de maneira a que se obtenham bons resultados num curto período.

Com este projeto aprendemos, para diferentes situações, o modo como se pode verificar a semelhança entre conjuntos implementando vários métodos minHash para diferentes problemas. Deste modo, pudemos retirar a conclusão de que diversas aplicações desenvolvidas atualmente utilizam todas estas implementações para a resolução de imensos problemas da atualidade relativamente à área da ciência da tecnologia.

# Fontes

<https://mccormickml.com/2015/06/12/minhash-tutorial-with-python-code/>