Métodos Probabilísticos para Engenharia Informática
Ano Letivo: 2020/2021
Turma P4
Tomé Lopes Carvalho - 97939
Gonçalo Fernandes Machado - 98359

Relatório PL4: Algoritmos Probabilísticos

O programa encontra-se dividido em 2 scripts. O script MPEI_P4_A_Dados gera a minhash table de de filmes vistos de cada utilizador de u.data e a minhash table de shingles dos nomes dos filmes de *u_item.txt*. O script MPEI_P4_A_Programa, com que o utilizador interage, contém as opções correspondentes às funcionalidades desejadas (listar filmes de um utilizador, mostrar sugestões, procurar filmes por título e terminar a execução da aplicação).

Funcionamento de MPEI_P4_A_Dados

1. Carregar os dados úteis (primeiras 2 colunas) de *u.data* para a variável *u_data*;
2. Carregar os dados de *u_item.txt* para a o cell array *u_item*;
3. Através de *u_data*, obter a lista de utilizadores;
4. Para cada utilizador, determinar os filmes que viu e guardar essa informação no cell array *Set*;
5. Para cada conjunto de filmes visto pelos utilizadores, gerar a linha respectiva da minhash table utilizando a hash function *DJB31MA*;
6. Guardar no cell array movies a lista de nomes dos filmes;
7. Para cada filme, gerar a linha respectiva da minhash table, como no passo 5;
8. Guardar as variáveis *u_data*, *u_item*, *min_hash_table*, *min_hash_table_shingles*, *k* e *K* num ficheiro.

Funcionamento de MPEI_P4_A_Programa

1. Carregar os dados presentes em MPEI_P4_data3.mat, onde estão presentes as minhash tables dos utilizadores e dos títulos dos filmes, *k* (tamanho de cada shingle), *K* (número de hash functions) e os dados em *u_item* e *u_data*;
2. Inicializar a variável *u_id*, onde se irá guardar o valor do ID do user, a 0 pois é um valor inválido;
3. Usando um ciclo while, pedir o ID do user até ser um id válido;
4. Inicializar a variável *opt*, onde se irá guardar o valor da opção escolhida pelo utilizador, como 0 pois é um valor inválido;
5. Usando um ciclo while e enquanto *opt* não tiver o valor 4:
    Imprimir as opções possíveis;
    Registar a opção do utilizador;
    Utilizando switch case com argumento *opt*, chamar as funções *print_user_movies*, *print_suggestions* e *search_movies* para os casos 1, 2 e 3 respetivamente, para o caso 4 utilizar um break para terminar a aplicação e em qualquer outro caso avisar o utilizador que a opção é inválida.

Função shingles(str, k)

1. Inicializar um cell array *shingle_set*, que será usado como set;
2. Percorrer os blocos de *k* (2º argumento) caracteres consequentes existentes na string passada como primeiro argumento e adicioná-los ao *shingle_set*;
3. Remover os elementos duplicados do cell array *shingle_set*, para que se "comporte" como um set.

Opção 1: função print_user_movies(u_id, u_data, u_item)

1. Encontrar todos os filmes do utilizador com id *u_id* presentes em *u_data* e guardar no cell array reviews;
2. Para cada filme, usar o ID para buscar o nome do filme em *u_item* e imprimir o nome;

Opção 2: função print_suggestions(u_id, u_data, u_item, min_hash_table)

1. Imprime todas as opções de categorias existentes;
2. Inicializa a variável *category* como inválida, para poder ser feita a verificação da variável introduzido pelo user;
3. Utilizando um ciclo while, pedir a categoria desejada até ser uma categoria válida;
4. Encontrar todos os filmes do utilizador com id *u_id* presentes em *u_data* e guardar no cell array *reviews_user*;
5. Inicializar a variável *K* com valor 200 que é o número de hash functions;
6. Inicializar a variável *dist_similar_user* com valor 2. Esta variável irá ser utilizada para guardar o valor da distância de Jaccard do user mais similar ao utilizador com id *u_id*;
7. Para cada utilizador presente em *u_data* exceto o utilizador atual, calcular a distância de Jaccard e comparar com *dist_similar_user*. Caso seja menor, redefinir *dist_similar_user* e definir most_similar_user com o respectivo ID.
8. Encontrar todos os filmes que o user mais similar já viu e guardar em *reviews_similar*;
9. Para cada filme visto por *most_similar_user*, buscar o ID do filme e verificar se esse filme pertence à categoria escolhida e se o utilizador atual não o viu, sendo que em caso verdadeiro é procurado o nome do filme e posteriormente impresso.

Opção 3: função search_movies(u_item, min_hash_table_shingles, k, K)

1. Inicializa a variável *str* como inválida, para poder ser feita a verificação da variável introduzido pelo user;
2. Utilizando um ciclo while, pedir uma string (*str*) ao utilizador até ser de tamanho válido ;
3. Obter o conjunto de shingles de *str* através da função shingles;
4. Inicializar a minhash table de shingles (com valor infinito);
5. Para cada shingle no conjunto, criar um array de hash codes e atualizar a minhash table de shingles do conjunto;
6. Criar um array coluna de estimativas de distâncias de Jaccard, *d_arr*, inicializado a zeros;
7. Preencher *d_arr* com as distâncias estimadas aos títulos dos outros filmes através de minhash;
8. Adicionar uma segunda coluna a d_arr, com os índices;
9. Ordenar *d_arr* por ordem crescente com base na primeira coluna (distâncias), utilizando a função *sortrows*;
10. Guardar na variável *top5* as primeiras 5 linhas de *d_arr*, isto é, as distâncias aos 5 títulos mais próximos e os respectivos índices dos filmes;
11. Imprimir “Top 5 matches and their Jaccard distance estimate:”;
12. Inicializar *matches_shown*, um contador de resultados imprimidos, a 0;
13. Para cada filme nos top 5 resultados:
    - Se a distância for igual ou inferior a 0.99:
        - Buscar o ID do filme, *movie_id*, para, de seguida, buscar o nome do filme;
        - Imprimir o nome do filme e a estimativa da distância de Jaccard;
        - Incrementar *matches_shown*;
    - Se a distância for superior a 0.99
        - Se não tiver sido mostrado nenhum resultado, imprimir “No matches found”
        - Caso contrário, imprimir “No additional matches found.”

Nota: Como *top5* está ordenado por ordem crescente de distância, se um resultado tem distância > 0.99, o resultado seguinte tem, obrigatoriamente, uma distância igual ou superior, pelo que não é necessário continuar o ciclo; por isso, faz-se break;

Escolha de *K* (número de hash functions)

Decidimos utilizar 200 hash functions, uma vez que, ao realizar a secção 4.3 do guião prático, foi o número de hash functions que levou a melhores aproximações da distância de Jaccard. Uma vez que os dados são gerados apenas uma vez e que gerar o conjunto de shingles na opção 3 é uma operação relativamente rápida, a velocidade do programa (MPEI_P4_A_Programa) não é afetada significativamente.

Escolha de *k* (tamanho de cada shingle)

Para decidirmos o tamanho de cada shingle realizamos uma série de testes. Estes testes consistiram em, para cada tamanho de shingles de 2 a 5, pesquisarmos certas palavras representativas da maioria dos casos e ver o quão acertadas eram as sugestões apresentadas. Enviamos no zip os dados gerados com k de 2 a 5. O nome do ficheiro indica o *k* utilizado, por exemplo no ficheiro MPEI_P4_data3.mat é utilizado *k* = 3.

Conclusões obtidas para cada *k*

k	Conclusões
2	Possível procurar por palavras de 2 letras (raras), muitos resultados são consequentemente pouco relevantes
3	Possível procurar por palavras chave de 3 letras, alguns resultados pouco relevantes
4	Resultados mais estritos, não possibilita procura por palavras de 3 letras, que são algo comuns
5	Resultados muito restringidos, impossibilita a procura por palavras de 4 ou menos letras, que são muito comuns nos títulos dos filmes

Exemplo: pesquisa por “war hero” com *k* = 2

```
Select choice: 3
Write a string: war hero
Top 5 matches and their Jaccard distance estimate:
Mother (1996) - 7.850000e-01
Super Mario Bros. (1993) - 8.100000e-01
Panther (1995) - 8.100000e-01
Kull the Conqueror (1997) - 8.150000e-01
Kull the Conqueror (1997) - 8.150000e-01
```

Nenhum dos resultados parece relevante à pesquisa efetuada. Resultados como “Super Mario Bros. (1993)” aparecem apenas porque contêm shingles “er”, “ar”, “ro” e “r “. Por este motivo descartamos este valor de k.

Exemplo: pesquisa por “Dog” com *k* = 3

```
Select choice: 3
Write a string: Dog
Top 5 matches and their Jaccard distance estimate:
Wag the Dog (1997) - 9.350000e-01
Mad Dog Time (1996) - 9.550000e-01
Shaggy Dog, The (1959) - 9.600000e-01
All Dogs Go to Heaven 2 (1996) - 9.650000e-01
Reservoir Dogs (1992) - 9.650000e-01
```

Este exemplo demonstra a importância de ser possível pesquisar por palavras de 3 letras. Existem 3 filmes com “Dog” no título e 2 com “Dogs”. Este é um exemplo entre muitos, pois existem muitas palavras com 3 letras, ao contrário de apenas 2. Com *k* > 3 não seria possível encontrar nenhum destes filmes se o utilizador se lembrasse que o filme que pretende ver tem “Dog” no nome, como no seguinte exemplo (*k* = 4):

```
Select choice: 3
Write a string: Dog
Top 5 matches and their Jaccard distance estimate:
No matches found.
1 - Your Movies
2 - Get Suggestions
3 - Search Title
4 - Exit
```

1608	Volcano (1997)
1609	Wag the Dog (1997)
1610	Waiting for Guffman (1996)
1611	Waiting to Exhale (1995)
1612	Walk in the Clouds, A (1995)
1613	Walk in the Sun, A (1945)
1614	Walkabout (1971)

No entanto, *k* = 4 tem o benefício de mostrar menos resultados irrelevantes, como “Shawshank Redemption, The (1994)” no exemplo seguinte (*k* = 3):

```
Select choice: 3
Write a string: Jaws
Top 5 matches and their Jaccard distance estimate:
Jaws (1975) - 7.950000e-01
Jaws 2 (1978) - 8.400000e-01
Jaws 3-D (1983) - 9.050000e-01
Shawshank Redemption, The (1994) - 9.700000e-01
No additional matches found.
```

Exemplo anterior com *k* = 4:

```
Write a string: Jaws
Top 5 matches and their Jaccard distance estimate:
Jaws (1975) - 8.650000e-01
Jaws 2 (1978) - 8.750000e-01
Jaws 3-D (1983) - 9.100000e-01
No additional matches found.
```

Para k = 5, é impossível procurar através de uma palavra de 4 ou menos letras. Sendo que existe uma grande quantidade de palavras desta dimensão nos títulos dos filmes, rapidamente descartamos este possível valor para k.

A decisão final foi entre k = 3 e k = 4.
Por um lado, k = 3 permite-nos pesquisar palavras de 3 letras, que são comuns nos títulos dos filmes, perdendo no entanto um pouco de relevância das sugestões. Por outro lado, k = 4 torna as sugestões mais relevantes, tendo a desvantagem de não se poder procurar por palavras com 3 letras.
Concluímos que compensa a perda de relevância de k = 3 em troca da possibilidade de procurar por palavras de 3 letras.