Universidade de Aveiro

Projeto de Métodos Probabilísticos para Engenharia Informática

Métodos Probabilísticos para Engenharia Informática

Cristiana Carvalho, Daniela Simões

Mestrado Integrado em Engenharia de Computadores e Telemática

Projeto de Métodos Probabilísticos para Engenharia Informática

Cristiana Carvalho, nº77682 Daniela Simões, nº76771

Docente: Paulo Monteiro

2015/2016

Conteúdo

Capítulo 1

Introdução

Neste trabalho pretende-se que sejam aplicados os conhecimentos adquiridos na cadeira de Métodos Probabílisticos para Engenharia Informática. Os conceitos a aplicar são Bloom Filter, Min-Hashing e Distância de Jaccard, que serão explicados posteriormente com mais detalhe. Essencialmente, os testes aplicados consistem numa biblioteca onde o utilizador solicita um livro, nesta fase inicial, o Bloom Filter é responsável por informar o utilizador se o livro existe ou não na Biblioteca. Uma vez que é efetuada a requisição do livro, o Min-Hash fica responsável por dar a conhecer ao utilizador outros utilizadores com escolhas semelhantes (através também da distância de Jaccard), e o Bloom Filter fica responsável por dar a conhecer ao utilizador opções semelhantes à sua requisição. Este exemplo foi escolhido devido à aproximação à vida real, e também por ser um exemplo onde são aplicáveis os conceitos essenciais a serem avaliados neste projeto.

Capítulo 2

Bloom Filter

O Bloom Filter é uma estrutura de dados eficiente, utilizado para testar se determinado elemento pertence ou não a um conjunto. No entanto, este método traz algumas desvantagens, nomeadamente o facto de poderem existir falsos positivos. Ou seja, por vezes o resultado obtido indica que o elemento existe no conjunto, quando isso efetivamente não acontece. Contudo, nunca existem falsos negativos, isto é, quando o resultado indica que não existe determinado elemento no conjunto, é porque realmente isso acontece.

2.1 K ótimo

No presente projeto, pode também otimizar-se o Bloom Filter, através do uso de um adequado K. Isto é, a quantidade de vezes que é necessário operar para se obter um resultado mais fidedigno. O K ótimo pode ser calculado, experimentalmente através da expressão:

m = ceil((n * log(p)) / log(1.0 / (pow(2.0, log(2.0)))));
k = round(log(2.0) * m / n);

- n - número de elementos a adicionar ao Bloom Filter
- m - número de bits no Bloom Filter
- p - probabilidade de falsos positivos
- k - número de HashFunctions

2.2 Falsos Positivos

2.3 Testes

Para os testes, utilizaram-se alguns exercícios propostos nos guiões realizados nas aulas.

2.3.1 Teste 1

De forma a testar as funções *insert()* e *isMember()* utilizou-se o seguinte código:

```
X = initialize(15);
Cidades = {'Aveiro';'Agueda'};

for i=1:length(Cidades)
  X = insert(X, Cidades{i}, 3);
end

isMember(X, 'Agueda', 3)
isMember(X, 'Oronhe', 3)
isMember(X, 'Aveiro', 3)
```

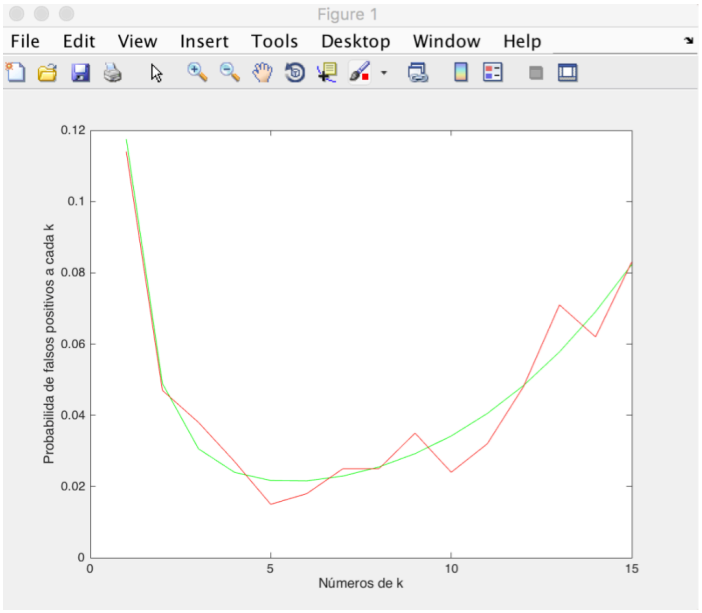
O resultado obtido foi o seguinte:

```
ans =

     1


ans =

     0


ans =

     1
```

2.3.2 Teste 2

Relativamente aos falsos positivos, para k=15, calcularam-se os valores teoricos para cada k, correspondente à linha verde apresentada no gráfico abaixo.

Calcularam-se ainda os valores experimentais para cada k, correspondente à linha vermelha apresentada no gráfico abaixo. Obtiveram-se os seguintes resultados:



Capítulo 3

Similaridade de Jaccard

O coeficiente de similaridade de Jaccard é utilizado para comparar conjuntos de amostras, calculando a semelhança entre os mesmos. Esta similaridade é calculada através da seguinte equação:

$$J(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|} \tag{3.1}$$

Capítulo 4

Distância de Jaccard

A distância de Jaccard é utilizado para comparar as diferenças de conjuntos de amostras. Esta distância é calculada através da seguinte equação:

$$J(A,B) = 1 - \frac{|A \bigcap B|}{|A \bigcup B|} \tag{4.1}$$

4.1 Testes

Para testes, utilizaram-se alguns exercícios propostos nos guiões realizados nas aulas. Relativamente à distância de Jaccard, utilizando o ficheiro 'u.data' - que contém essencialmente ID's de filmes e utilizadores - compararam-se os filmes que cada utilizador via. Utilizando a equação apresentada acima calculou-se a distância de Jaccard, e assumiram-se como semelhantes todos aqueles que tivessem a distância menor que 0.4. Calcularam-se ainda os tempos de execução do cálculo das distâncias e dos similares, respetivamente. Obtiveram-se os seguintes resultados:

```
Elapsed time is 77.210646 seconds.
Elapsed time is 0.023505 seconds.

SimilarUsers =

  328.0000  788.0000    0.3270
  408.0000  898.0000    0.1613
  489.0000  587.0000    0.3701
```

Capítulo 5

Funções de Hashing

No presente projeto são usadas duas funções de hashing diferentes.

5.1 Decisões

Foi decidido usar duas funções de hashing diferentes, pois foi implementada a Universal Hashing, que se considera funcionar bastante bem para inteiros, no entanto, quando se tratam de strings, o processo torna-se demasiado lento, o que não é o pretendido. Posto isto foram implementadas outras três funções de hashing para strings, uma com o uso de sementes, outra com a concatenação de carateres e outra com a conversão de string para inteiros. Optou-se por usar a última por questões de eficiência, no entanto as outras podem ser consultadas no diretório do projeto.

Capítulo 6

Min-Hashing

Min-Hasing é uma técnica utilizada para testar rapidamente se dois conjuntos são ou não semelhantes. Neste projeto, a técnica usada foi a seguinte:

- Usar uma de duas hashfunctions: uma HashFunction que opere sobre inteiros no caso da base de dados a ser lidas conter os IDs dos livros, ou uma HashFunction que opere sobre strings no caso da base de dados a ser lida contiver os títulos dos livros.
- Gerar K valores através da HashFunction e escolher o mínimo.
- Guardar os mínimos gerados num array.
- Contar todos os mínimos que sejam iguais, e dividir pelas K iterações.
- Para o cálculo da distância de Jaccard, como já especificado, é necessário fazer: 1 - (valor calculado na alínea anterior).

  Após este processo, consideram-se similares os livros cujo valor da última operação tenha um valor menor que 0.4.

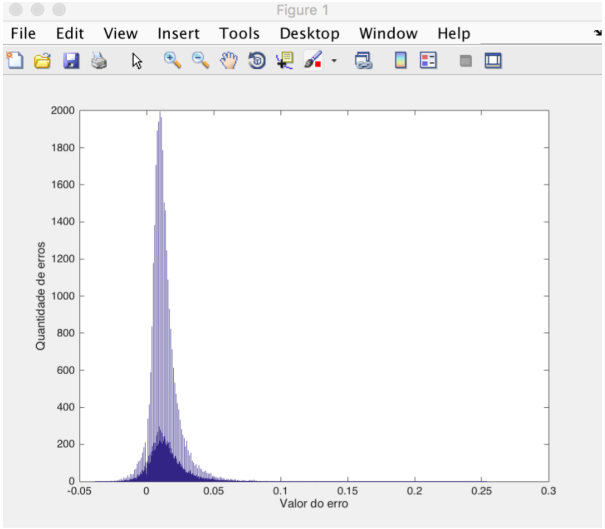
Usando primeiramente o exemplo da aula, obtivémos como resultado através do Min-Hash:

```
SimilarUsersMinHash =

  328.0000  788.0000    0.3330
  408.0000  898.0000    0.1490
  489.0000  587.0000    0.3710
```

Capítulo 7

Distância de Jaccard Teórica vs Min-Hash

Neste capítulo será apresentado um gráfico que mostra o erro entre o cálculo da distância de Jaccard pela fórmula teórica e o cálculo do mesmo através de Min-Hash.



Através deste gráfico é possível retirar várias conclusões:

- Os erros apresentados têm uma média bastante baixa, significando assim que os valores não variam muito e portanto o erro não é significativo;

- O gráfico apresenta um **Distribuição Gaussiana** - ou distribuição normal. Ou seja, pode-se saber os valores da média e do desvio padrão. Conhecendo estes valores, é possível determinar qualquer probabilidade.
- Obtem-se uma variância de cerca de **2.6276e-04**, que por ser bastante baixa, indica que os valores se encontram próximo do valor esperado.

Capítulo 8

Aplicação

A aplicação desenvolvida, como já referido, é uma biblioteca virtual. Inicialmente, o utilizador terá de inserir o seu número de utilizador e o seu título. Assim que o mesmo for inserido, ser-lhe-à apresentado uma de duas coisas: ou o livro existe na biblioteca, e o seu número de utilizador e livro requisitado são adicionados à base de dados ou então o livro não existe e a requisição não pode ser concluída. Foi criada também uma opção de pesquisar pelo ID do livro, numa base de dados onde tenha ID do utilizador e ID do livro. Esta vertente permite saber tanto quanto a já referida, simplesmente faz uso do ID do livro e não do seu título.

```
Trial>> Library
Qual o seu ID de utilizador? 76771
Qual o título do livro a requisitar? 'O Cancioneiro portuguez da Vaticana'
O seu livro possivelmente existe!
```

Em qualquer das opções, acha-se uma boa medida mostrar ao utilizador outros livros que possa gostar, isto é: Se o livro não existir, faz-se uma listagem de todos os livros existentes para que o utilizador tenha total liberdade de escolha. Já para os utilizadores que conseguem requisitar o seu livro, serão apresentados outros livros semelhantes que existam na biblioteca. Para o livro ser semelhante é necessário que tenha pelo menos metade das palavras iguais. Considera-se uma medida útil visto que pode despertar interesse ao utilizador a requisitar novamente na biblioteca.

```
Qual o seu ID de utilizador? 21
Qual o título do livro a requisitar? 'Livro 1'
O seu livro possivelmente existe!
Outras sugestões: Livro 2.txt
```

8.0.1 Passos de utilização

1. Correr o ficheiro **Library.m**;
2. Responder às questões colocadas como é mostrado nos exemplos anteriores;
3. A partir daí toda a informação será disponibilizada automaticamente.

8.1 Decisões

Optou-se por usar o Bloom Filter para a apresentação de livros semelhantes, pois considerou-se que a utilização de Shingles não se aplicava à aplicação em causa. O que se quer é oferecer ao utilizador hipóteses de leitura que se enquadrem, provavelmente, com os seus gostos. Ao utilizar o Bloom Filter consegue-se então perceber quais os livros que incidem mais sobre determinadas palavras, ao inverso da utilização de Shingles, que por operar sobre cadeias de carateres será mais adequada para detetar ficheiros-cópia, e não ficheiros semelhantes. Optamos por não analisar o gráfico de erro da nossa aplicação que por ter uma base de dados pequena não é percetível a sua forma Gaussiana. No entanto já analisámos um anteriormente que por ter uma base de dados considerável é bastante percetivel a sua forma.

8.2 Testes

Foram realizados diversos testes em relação à aplicação desenvolvida. Até agora demonstrou-se que os módulos desenvolvidos correspondiam aos enunciados pedidos, agora ir-se-à motsrar o desempenho face à aplicação presente. Os livros que constam na biblioteca virtual são, "Os Lusíadas", "O Cancioneiro portguez da Vaticana", "O Congresso de Roma".

Source: Gutenberg

8.2.1 Fase Inicial

```
Qual o seu ID de utilizador? 76771
Qual o título do livro a requisitar? 'O Cancioneiro portuguez da Vaticana'
O seu livro possivelmente existe!
Outras sugestões: Os Lusiadas.txt
Elapsed time is 0.512518 seconds.
Elapsed time is 0.001683 seconds.
```

Posto isto, é escrito no ficheiro books.data:

```
76771   O Cancioneiro portuguez da Vaticana.txt
76771   O Congresso de Roma.txt
67405   O Cancioneiro portuguez da Vaticana.txt
67405   Os Lusiadas.txt
67405   O Cancioneiro portuguez da Vaticana.txt
```

Após ser escrito no ficheiro, pode então calcular-se a similaridade entre utilizadores. *a distância de Jaccard* entre o utilizador 76771 e 67405 neste caso é de **0.1304**.

Para outra base de dados, por exemplo, obtem-se:

```
SimilarUsers =

  743.0000  745.0000    0.1739
```

```
SimilarUsersMinHash =

  743.0000  745.0000    0.3270
```

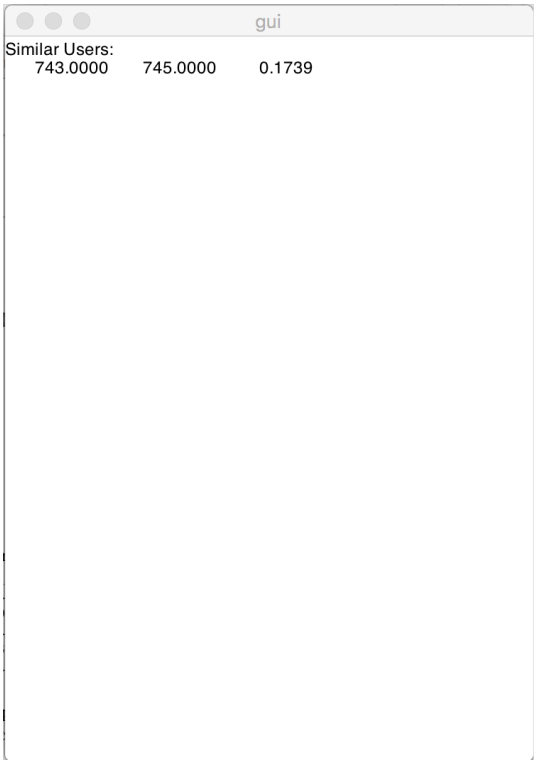


Figura 8.1: Similar Users em interface gráfica

8.2.2 K-Ótimo

Para a aplicacção presente, o K-Ótimo determinado através da expressão escrita anteriormente é de K = 20.

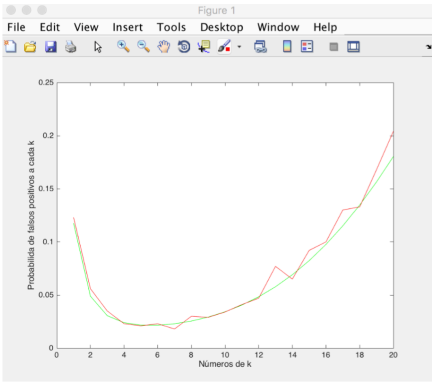


Figura 8.2: Probabilidade de Falsos Positivos

8.2.3 HashString

```
1 classdef HashFunctionR < handle
2 %
  ------------------------------------------------------------------------

3 % Class:          HashFunction < handle
4 %
5 % Constructor:    Hf = HashFunction([m]);
```

```
6  %
7  % Properties:    (none)
8  %
9  % Methods:       value  = Hf.HashCode(key);
10 %
11 % Indexing:      (no indexing supported)
12 %
13 % Description:  This class implements an universal
       hash function for
14 %                alphanumeric character keys.
15 %                Adapted from
16 %                    http://www.mathworks.com/
       matlabcentral/fileexchange/45123-data-structures/
       content/Data%20Structures/Hash%20Tables/HashTable.m
17 %
18 % Author:        Brian Moore          edited by
       Ricardo Jesus
19 %                brimoor@umich.edu
       ricardojesus at ua dot pt
20 %
21 % Date:          January 16, 2014
22 %
   ------------------------------------------------------------------------------

23
24      %
25      % Private properties
26      %
27      properties (Access = private)
28          m;                  % Maximum hash value
29          p;                  % hash function parameter
30          a;                  % hash function parameter
31          b;                  % hash function parameter
32          c;                  % hash function parameter
33      end
34
35      methods
36          %
37          % Initialize hash function
38          %
```

```
39          % For HashCode() to be universal, the
                following criteria should be
40          % met:
41          %
42          % - Criteria for this.p -
43          %   1) this.p is prime
44          %   2) this.p > this.m
45          %   3) this.p > 75 = #{possible values (
                alphanumeric) for key(i)}
46          %   3) this.p is large compared to
                ExpectedValue(length(key))
47          %
48          % - Criteria for randomized parameters -
49          %   1) this.a is an integer in [1, ..., this.p
                - 1]
50          %   2) this.b is an integer in [0, ..., this.p
                - 1]
51          %   3) this.c is an integer in [1, ..., this.p
                - 1]
52          %
53          function self = HashFunctionR(m)
54              % Set m parameter
55              if exist('m', 'var')
56                  self.m = m;
57              else
58                  self.m = 2^64-1;
59              end
60              % Set prime parameter
61              ff = 1000;  % fudge factor
62              pp = ff * max(self.m + 1,76);
63              pp = pp + ~mod(pp,2);  % make odd
64              while (isprime(pp) == false)
65                  pp = pp + 2;
66              end
67              self.p = pp;  % sufficiently large prime
                    number
68
69              % Randomized parameters
70              self.a = randi([1,(pp - 1)]);
71              self.b = randi([0,(pp - 1)]);
```

```
72              self.c = randi([1,(pp - 1)]);
73          end
74
75          %
76          % Compute the hash code of a given key
77          %
78          % - Assumptions for key -
79          %   1) key(i) is alphanumeric char: {[0...9],
                [a...z], [A...Z]}
80          %
81          function hk = HashCode(self, key)
82              % Convert character array to integer array
83              ll = length(key);
84              if ~ischar(key)
85                  error('Keys must be nonempty
                        alphanumeric strings');
86              end
87              key = double(key) - 47;  % key(i) =
                    [1,...,75]
88
89              %
90              % Compute hash of integer vector
91              %
92              % Reference: http://en.wikipedia.org/wiki/
                    Universal_hashing
93              %            Sections: Hashing integers
94              %                      Hashing strings
95              %
96              hk = key(1);
97              for i = 2:ll
98                  % Could be implemented more
                        efficiently in practice via bit
99                  % shifts (see reference)
100                 hk = mod(self.c * hk + key(i), self.p)
                        ;
101             end
102             hk = mod(mod(self.a * hk + self.b, self.p)
                    , self.m) + 1;
103         end
104     end
```

```
105
106 end
```

E é usada com Universal Hashing:

str = HF.HashCode(Setij);

$hash_code = mod(mod(FirstRand(k) * str + SecondRand(k), prime), 1000000005721)$

8.3 Conclusão

Em suma, considera-se que este trabalho foi útil na medida a consolidar conhecimentos adquiridos, assim como a perceber a utilidade aplicada à vida real. A aplicação desenvolvida como visto é uma biblioteca virtual que requer apenas que sejam inseridos o ID do utilizador e o título do livro. Podendo ser utilizado strings ou inteiros, a opção foi mostrar testes com strings, visto que os módulos já foram testados anteriormente com valores inteiros, então tentou-se evitar ser redundante.

Bibliografia

[1] ***K-ótimo***
http://hur.st/bloomfilter?n=4p=1.0E-6