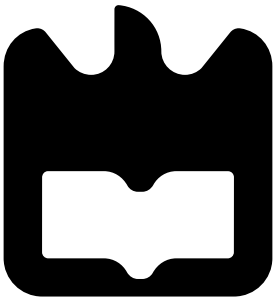
Book Management

Ana Tavares (N.Mec 76629)
Diogo Ferreira (N.Mec 76425)



Conteúdo

Capítulo 1

Introdução

No âmbito do trabalho proposto, este relatório consiste na explicação, demonstração e teste dos módulos necessários ao mesmo : Bloom Filter para avaliar pertença a um conjunto e MinHash para a descoberta de items similares.

Será também descrita a aplicação desses mesmos módulos a um Dataset de book-crossing criado por integrantes da Universidade de Friburgo.

Capítulo 2

Módulos

2.1 Bloom Filter

Um Bloom Filter é uma estrutura probabilistica e eficiente em termos de espaço criada por por Burton Howard Bloom, em 1970, cujo objectivo é testar a pertença de um membro a um dado conjunto. Uma vez que se trata de uma estrutura probabilistica, é frequente acontecerem alguns falsos positivos , no entanto é garantido que não hajam falsos negativos, devolvendo uma resposta de "possivelmente pertence"ou "definitivamente não pertence". Os elementos podem ser adicionados ao conjunto mas não podem ser removidos, uma vez que esta estrutura baseia-se numa mera anotação de existência. Quanto maior o set adicionado ao Bloom Filter, maior será a probabilidade de falsos positivos.

Os módulos constintuintes deste Algoritmo são os seguintes:

- **BloomFilter.m** - Surge como uma automatização do processo de criação e inserção do filtro, inicializando-o e invocando o módulo insert.m para a inserção do set.
- **insert.m** - Trata da inserção dos elementos do Set no bloom, criando também as k Hash Functions.
- **isMember.m** - Avalia a pertença de um elemento ao conjunto, analisando simplesmente se os valores obtidos através da sua Hash Function já têm uma entrada no filtro.

2.1.1 Hash Function

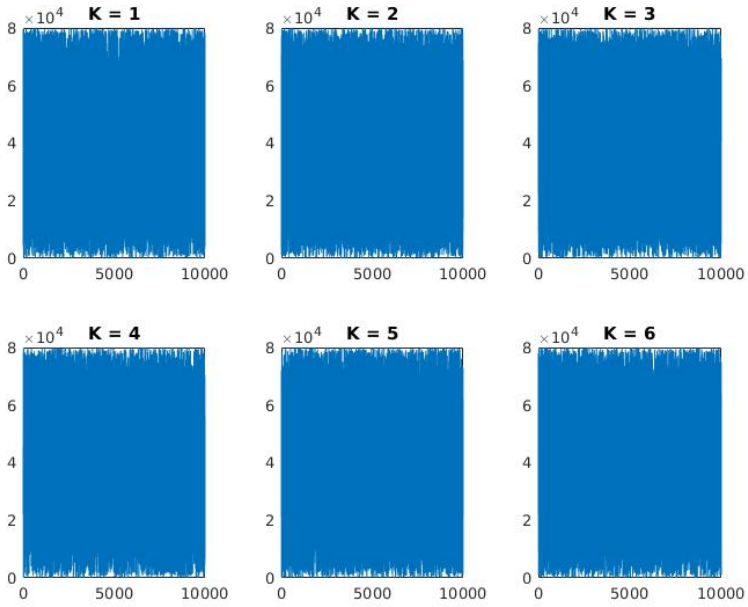
O Bloom Filter utiliza Hash Functions para fazer a indexagem de um set de dados para o filtro , sendo que a pertença de um membro é testada simplesmente verificando o resultado da Hash Function para esse membro relativamente ao filtro existente.

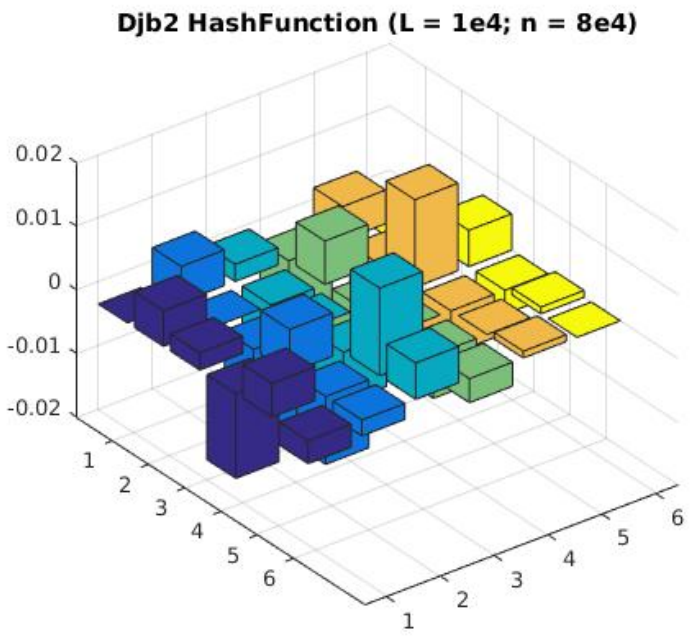
Como tal, uma pesquisa foi realizada com o objectivo de determinar qual a Hash Function que melhor se enquadraria no nosso objecto de estudo, sendo

que duas Hash Functions escolhidas foram devidamente testadas no que toca aos valores de correlação através do módulo **TestHashFunction.m** , concluindo que a Djb2 foi a Hash Function que apresentou melhores resultados e portanto foi a adoptada para este nosso trabalho.

Dbj2 Hash Function

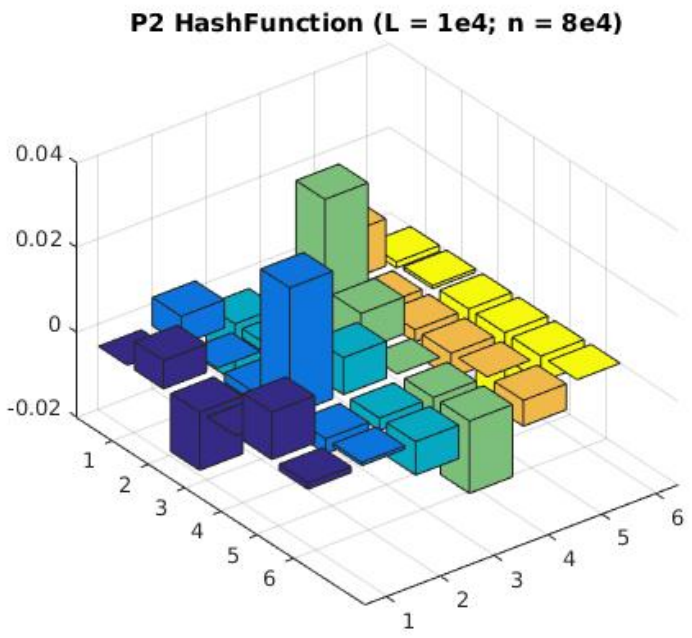
Este algoritmo de Hashing foi "criado"e publicado por Dan Bernstein no grupo comp.lang.c. Após a sua criação, novas versões foram sendo lançadas, ainda que a original se mantenha como a mais utilizada. Há um grande mistério por trás das constantes utilizadas (33 e 5318) , sendo este constantemente relacionado com o Método da Congruência (LCG).





P2 Hash Function

Esta Hash Function foi fornecida pelos Docentes de Programação II da Universidade de Aveiro e, ainda que apresente resultados relativamente bons, não conseguiu superar a Djb2, sendo utilizada no âmbito de trabalho para mera comparação de performance.



2.1.2 Testes

Com o objectivo de assegurar o correcto funcionamento deste módulo, foram realizados os seguintes testes:

BasicTest.m

Este teste baseia-se na existência de dois Sets com nomes próprios, introduzindo o primeiro no Bloom Filter e testando se algum dos membros do segundo é membro do filtro . Este pequeno teste calcula ainda Probabilidade teórica de Falsos Positivos , comparando-a posteriormente com a Probabilidade prática obtida.

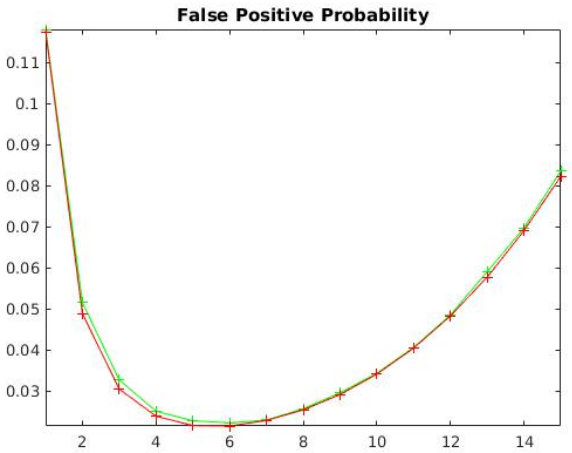
```
*** Inserted Set ***
 Filipa is Probably a Member
 Leticia is Probably a Member
 Ana is Probably a Member
 Rita is Probably a Member

*** Another Set ***
 Diogo is Not a Member
 leticia is Not a Member
 Luis is Not a Member
 Ana is Probably a Member

False Positives (T) = 0.00144595
False Positives (P) = 0
```

RandomStringTest.m

Este teste um pouco mais complexo gera dois Sets de strings aleatórias diferentes (criados através do módulo **RandomStringGenerator.m**), calculando posteriormente os valores teóricos (a vermelho) e práticos (a verde) dos falsos positivos para um dado n (tamanho do Filtro) e um dado L (tamanho do Set) introduzidos, com uma dada gama de k (numero de Hash Functions, com $1 \leq k \leq 15$) , apresentando posteriormente um gráfico com ambas as curvas sobrepostas para uma melhor comparação.



TestWithOptimalN.m

Com um dado L e p (probabilidade de falsos positivos que estamos dipostos a correr) , é possivel determinar analiticamente o valor de n (tamanho do Bloom Filter) que minimiza espaço e recursos computacionais com base na seguinte expressão:

$$n = \frac{L \cdot \log \frac{1}{p}}{\log 2^2}$$

Este módulo trata de calcular o n e k óptimo(explicitado no módulo seguinte), com base na geração de Strings aleatórias, apresentando depois todas as conclusões finais.

```
Set Length = 100000
Chosen Probabilty = 1.000000e-04
Optimal N = 1917012
Optimal K = 13
```

TestWithOptimalK.m

Com um dado n e L é possivel calcular analiticamente o valor de K que minimiza a probabilidade de falsos positivos :

$$k = \frac{n}{L} \cdot \ln 2$$

Com base nesta fórmula analitica e nos valores de n e L, este módulo calcula o valor óptimo para K, tendo como objecto de utilização Strings aleatórias geradas pelo módulo RandomStringGenerator.m e com comprimento fixo, apresentando ainda a Probabilidade Teórica e Prática de Falsos Positivos.

```
Set Length = 100000
Filter  Length = 800000
Optimal K = 6
False Positive Probability(T) = 2.157714e-02
False Positive Probability(P) = 2.140000e-02
```

2.2 Similaridade de Jaccard

Dados 2 conjuntos A e B, a Similaridade de Jaccard, denotada por J(A,B), é calculada através da fórmula

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Consequentemente a distância de Jaccard é dada por:

$$dJ = 1 - J(A, B)$$

2.2.1 Testes

MoviesTest.m

Neste teste pretende-se avaliar a distância entre cada par de utilizadores (no total 943) que avaliaram um conjunto de filmes(no total 1682). O Dataset de 100K contendo os ratings foi retirado do site *"grouplens.org/datasets/movielens"*. Utilizando um limiar de distância de Jaccard menor do que 0.4 e usando a fórmula explicitada acima, obtiveram-se 3 pares de utilizadores similares com distância entre os seus conjuntos de filmes avaliados menor do que 0.4. Apresenta-se também o tempo de execução de 2 tarefas: cálculo da distância de Jaccard entre todos os pares e determinação dos pares com distância inferior ao limiar.

```
Trial>> Movies
threshold =
          0.4
SimilarUsers =
          328          788      0.32704
          408          898      0.16129
          489          587      0.37008
Elapsed time is 61.395708 seconds.
```

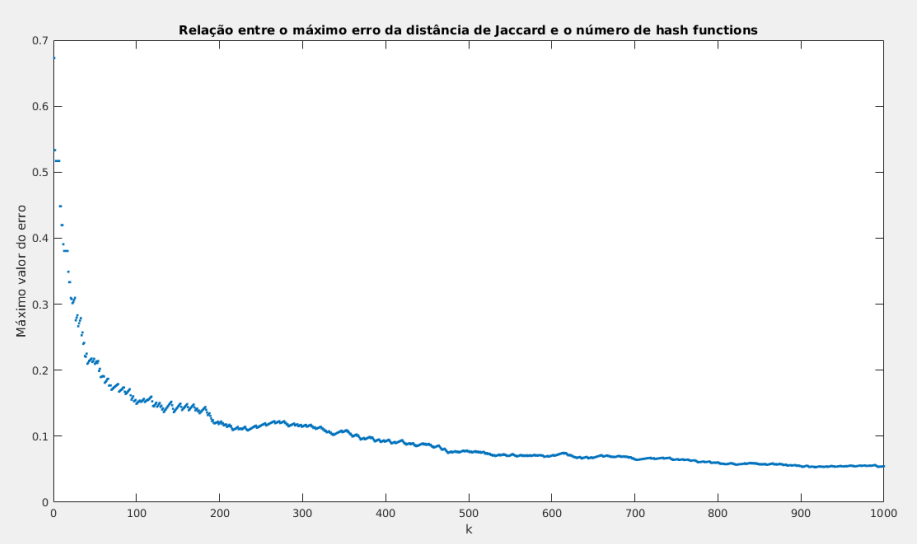
2.3 MinHash

A MinHash apresenta-se como uma técnica mais rápida para o cálculo da Similaridade e Distância de Jaccard. Assenta no príncipio da conversão dos conjuntos em assinaturas nas quais é preservada a similaridade entre os pares. Uma forma de obter as assinaturas é escolher k hashfunctions que mapeiam cada elemento dos conjuntos, cujo valor adicionado à assinatura será o menor obtido. Dados 2 conjuntos de 2 utilizadores(A,B) se o valor minimo mapeado for igual significa que o elemento da união $A \cup B$ com o menor valor encontra-se na intersecção $A \cap B$.

$Pr[hmin(A) = hmin(B)] = J(A, B)$. É por isso importante ter um número de hashfunctions k elevado, dado que $J(A, B)$ será igual ao quociente entre o nºde hashfunctions que de facto coincidem e k. A melhor forma de obter k hashfunctions diferentes é usando a Universal Hash Function: $h_{a,b} = ((a \cdot x + b) \bmod p) \bmod N$.

Os coeficientes a e b são gerados aleatoriamente na função **coef_a_b.m**. N designa a gama para qual os valores vão ser mapeados (é importante que seja bastante maior que o nº total de filmes para evitar colisões), e p é um número primo maior que N. Aplicando a técnica de MinHash ao exercício dos

filmes acima descrito era necessário escolher o nº de hashfunctions a usar(k). Calculando, no módulo **FindKMinhash.m**, para vários valores de k entre 1 e 1000, o máximo erro entre a distância de Jaccard calculada pela fórmula e utilizando a técnica de Minhash obteve-se o seguinte gráfico.



A matriz que contém os valores do máximo erro para cada k encontra-se no ficheiro **Dif_Matrix.mat**.
Pela análise do gráfico é evidente que o erro se torna constante para valores de k próximos de 1000.

Usando k igual a 1000 e mantendo o limiar de 0.4 obtiveram-se os mesmos 3 pares de utilizadores e distâncias bastante semelhantes relativamente às reais que foram apresentadas acima.

Apresenta-se também o tempo de execução de 2 tarefas do módulo **TestMinHash**: a execução do cálculo da distância de Jaccard entre todos os pares e determinação dos pares com distancia inferior ao limiar.É notória a redução significativa do tempo de 61.395708 segundos para 23.540005 segundos.

```
Trial>> TestMinHash
Elapsed time is 23.540005 seconds.

SimilarUsers =

  328.0000  788.0000    0.3310
  408.0000  898.0000    0.1660
  489.0000  587.0000    0.3830
```

Capítulo 3

Aplicação Conjunta

A demonstração conjunta dos módulos acima descritos será aplicada ao Dataset de Book-Crossing criado por integrantes da Universidade de Friburgo.Este Dataset é constituído por 3 ficheiros:

- **BX-Users** - Contém os ID's dos 278.858 utilizadores, o seu país e cidade(se for fornecida) e ainda a idade caso esta também seja fornecida no momento do registo.
- **BX-Books** - Contém 271.379 livros, os seus respetivos ISBN, título do livro, autor, ano de publicação e editora. Todos estes campos foram obtidos através da Amazon. Existem também 3 campos com links das imagens dos livros de 3 diferentes tamanhos. Alguns links das imagens estão indisponiveis, facto relacionado com a própria Amazon.
- **BX-Book-Ratings** - Contém as avaliações dos livros. Os ratings são explicitos (expressos numa escala de 1 a 10) ou implicitos (com valor 0).Considerámos ignorar os ratings com valor 0, pois achamos que estes não fornecem nenhuma informação relevante.

  Nota: Foi necessário alterar os documentos por questões de falha nos delimitadores dos campos, para que fosse possível fazer uma correta divisão dos dados.

Os dados destes ficheiros são lidos e processados através de dois módulos:

- **ReadData.m**- Trata da Separação dos Dados em 3 Sets, um relativo às informações dos Livros, outro a informações dos Utilizadores e outro às Avaliações efectuadas por cada Utilizador.
- **usersBooks.m**- Recebendo o Set contendo as avaliações efectuadas, cria um novo Set associando a cada utilizador o(s) livro(s) avaliados por si.

Decidimos criar uma interface gráfica para interação com o utilizador, pois consideramos ser mais dinâmico do que a mera execução dos programas. Toda a

lógica dos módulos será incorporada nesta interface, através do módulo **gui.m**. Uma versão de consola (sem inputs) está também disponivel através do módulo **withoutGUI** , ainda que este tenha servido meramente como fase inicial da aplicação.

A interface interligará os 2 módulos da seguinte forma:

3.1 Bloom Filter

Será utilizado para verificar se um dado livro existe, isto é, o utilizador pode usar a barra de procura para encontrar um dado livro. Aquando da introdução do nome e verificação da existência, uma mensagem será apresentada, mostrando não existência ou existência provável, devido às propriedades de um Bloom Filter. O Tamanho do Bloom Filter e do numero de HashFunctions foram calculados tendo em conta as fórmulas apresentadas na seccção anterior, aquando da explicitação deste módulo.

Como esta aplicação tem caracter apenas demonstrativo e de aprendizagem, apenas foram introduzidos no Bloom Filter 50.000 dos aproximadamente 271.000 livros disponiveis,com o objectivo de diminuir o tempo necessário para o processamento de informação e cálculos necessários.

3.2 MinHash

Será utilizada para encontrar utilizadores com avaliações semelhantes dentro de uma gama pequena, dado que temos 278.858 avaliações **positivas** (ratings acima de 5). Poderia também ser feito o inverso, isto é, encontrar aqueles que avaliaram de forma negativa os mesmos livros.

Em qualquer uma das situações encontram-se não só os utilizadores que avaliaram os mesmos filmes, mas também aqueles que partilham da mesma opinião, sendo esta postiva ou negativa.

3.3 Outras Funcionalidades

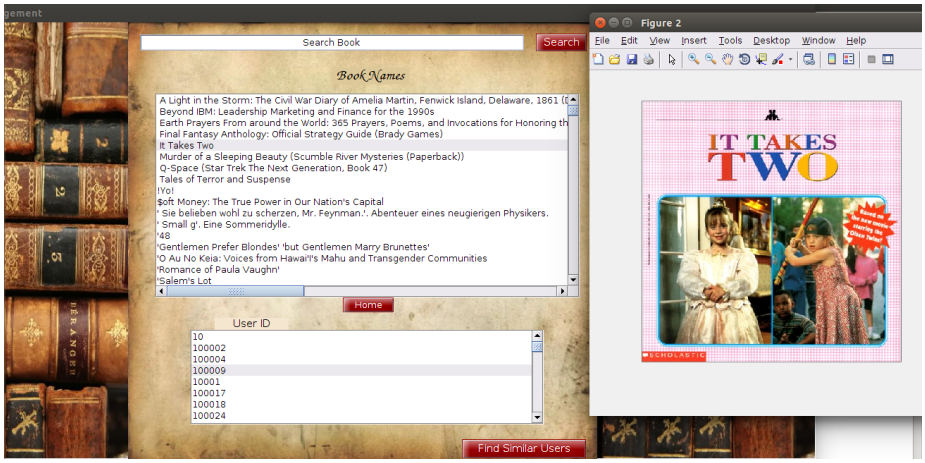
A interface gráfica é criada com base em 4 principais componentes:

3.3.1 Barra de Pesquisa

Permite-lhe verificar se um livro existe ou não no Dataset analisado, tendo em conta o caracter probabilistico associado a um Bloom filter. A pesquisa é feita de uma forma muito simples, bastando apenas introduzir o nome completo do filme (Case Sensitive) e clicando no botão *"Search"* onde, posteriormente, será apresentada uma mensagem com o resultado da pesquisa.

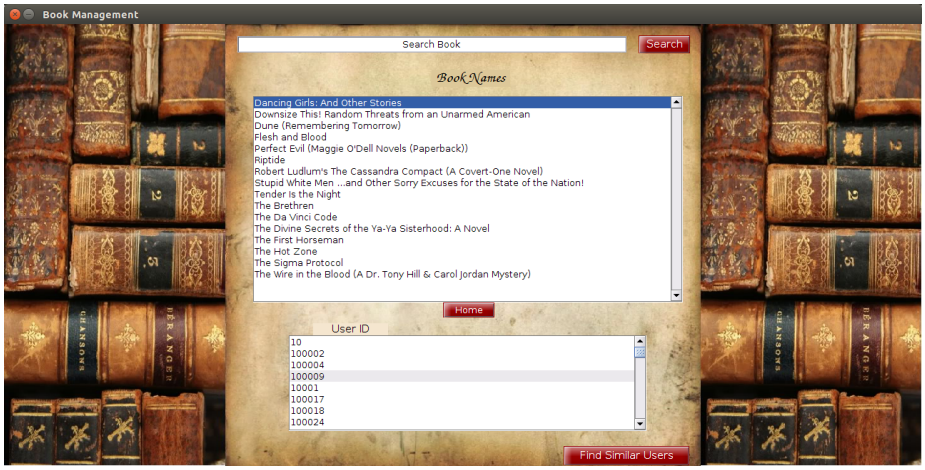
3.3.2 Lista de Filmes

Aqui poderá analisar os 271.000 filmes presentes no Dataset, com a possibilidade de ver a imagem da capa , aquando de um duplo clique em cima do seu nome. Caso a imagem da capa não exista ou não esteja disponivel , uma imagem afirmando isto mesmo será apresentada.



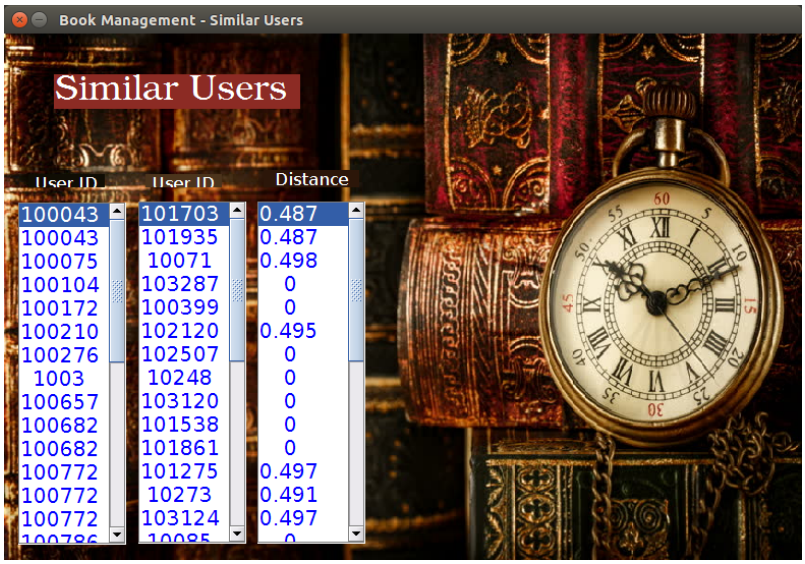
3.3.3 Lista de Utilizadores

Nesta lista poderá ver todos os 1000 utilizadores escolhidos de entre os 279.000 do Dataset. Para alem de apenas serem escolhidos 1000 de todo o conjunto possivel , pelas razões mencionadas anteriormente, estes também passaram por um processo de filtragem , por onde só passaram utilizadores que fizeram avaliações positivas, isto é, avaliações iguais ou superiores a 5. Esta filtragem tem fins meramente académicos e de diminuição do tempo de processamento. Terá ainda a possibilidade de , com um simples clique em cima do ID do utilizador, verificar quais os livros que este avaliou. Dado que o Dataset tem algumas avaliações correspondentes a livros inexistentes, poderá aparecer a mensagem *"*** Unavailable Books ***"* no caso de se tratar de uma destas.



3.3.4 Janela de Utilizadores Similares

Este componente pode ser acedido através do botão *"Find Similar Users"*, e tem o objectivo de detectar e apresentar Utilizadores cujas avaliações são iguais ou muito semelhantes, para que devidos ajustes sejam efectuados , no caso de uma aplicação mais real e prática. Esta Janela é constituida por 3 listas , sendo que nas duas primeiras poderá ver , horizontalmente, os ID's dos utilizadores com as avaliações semelhantes e, na terceira, a Distância de Jaccard entre eles. Uma vez que no Dataset existem muitos Utilizadores que apenas efectuaram uma unica avaliação, é muito frequente o aparecimento de Distâncias de Jaccard de 0.



Capítulo 4

Conclusão

Com a realização deste trabalho conseguimos verificar , de uma forma mais prática, que o mundo onde vivemos é , por definição, probabilistico, bem como conhecer algumas destas tecnologias probabilisticas utilizadas em produtos do nosso conhecimento. Este trabalho também desenvolveu as nossas capacidades e conhecimentos como programadores , principalmente no que toca a programação orientada ao cálculo numérico.

Bibliografia

[1] Bloom Filter - Wikipedia.
https://en.wikipedia.org/wiki/Bloom$_filter$

[2] MinHash Tutorial with Python.
https://chrisjmccormick.wordpress.com/2015/06/12/minhash-tutorial-with-python-code/