MPEI – Trabalho Prático Final

Relatório

Cristiano Vagos 65169
Ariel Bastos 72204

Data: 14/12/2015

Descrição do Trabalho Final

O nosso trabalho final passa pela utilização dos conceitos dados nos últimos guiões da disciplina, onde abordam os Bloom Filters, MinHash e Similaridade de Jaccard. O seguinte exemplo de aplicação foi o que nós escolhemos para demonstrar o uso conjunto de ambos os módulos, pelo que o passamos a descrever:

Imaginemos que estamos a visitar um website que reúne notícias (ex. Google News). Numa página referente a um artigo são mostrados outros artigos que o utilizador pode ver. O nosso objectivo é mostrar ao utilizador artigos semelhantes ao que ele está a ler. Para isso temos que:
- dado um conjunto de artigos ver se o utilizador já visitou o respectivo site (utilizando o Bloom Filter)
- analisar o texto de cada um dos artigos de forma a recomendar uma visita (Finding Similar Items)

Descrição dos Módulos

Como descrito acima, este trabalho contém os módulos BloomFilter e FindingSimilarItems, feitos para ser corrido no programa Octave (não temos Matlab instalado). Todo o código está devidamente comentado e pronto e funcionar. A descrição de cada uma das funções incluídas é a seguinte:

- **Módulo Bloom Filter**

Para a execução do Bloom Filter, foram criadas as seguintes funções:

- *BloomFilter (BloomFilter.m)*
Função que cria um BloomFilter (objecto).
Inputs: n (tamanho do BloomFilter) e k (número de funções hash a ser utilizadas).
Output: o objecto referente ao Bloom Filter.

- *insert (insert.m)*
Função que insere uma string num dado Bloom Filter.
Chama a função hash para calcular um valor de hash, e insere esse valor dentro do array do Bloom Filter de acordo com o número de funções hash dado pelo utilizador aquando da criação do Bloom Filter.
Inputs: bf (Bloom Filter criado anteriormente) e elem (string a ser inserida no Bloom Filter)
Output: obj (Bloom Filter com o novo elemento no array)

- *isMember (isMember.m)*
Função que verifica se uma dada string está (provavelmente) no array do Bloom Filter.

Chama a função hash para calcular um valor de hash e verifica se o valor hash está no array do Bloom Filter, de acordo com o número de funções hash dadas pelo utilizador aquando da criação do Bloom Filter.
Inputs: bf (Bloom Filter criado anteriormente) e elem (string a ser verificada no Bloom Filter)
Output: bool (resultado da operação – 0 (false) ou 1 (true))

- *string2hash (string2hash.m)*
Função hash que gera um valor hash a partir de uma string.
Retirada de http://www.mathworks.com/matlabcentral/fileexchange/27940-string2hash/content/string2hash.m
Inputs: str (string a ser gerada) e maxSize (tamanho máximo da hash a gerar)
Output: hash (valor hash gerado pela função – entre 0 e o tamanho máximo do array)

- *resultingHash (resultingHash.m)*
Função que calcula a hash calculada na função string2hash de acordo com o número de funções hash dadas pelo utilizador na criação do Bloom FIlter.
Inputs: hash (valor hash calculado na função string2hash), i (número da função hash actual) e arraySize (tamanho do array do Bloom Filter)
Output: result (resultado da operação)

- *getFalsePositiveProbability (getFalsePositiveProbability.m)* **[FUNÇÃO EXTRA]**
Função que calcula a probabilidade de falsos positivos num dado Bloom Filter.
Respeita a fórmula (1 - e^(-k * n / m)) ^ k
Inputs: bf (Bloom Filter criado anteriormente)
Output: value (valor da probabilidade)

- *clearBloomFilter (clearBloomFilter.m)* **[FUNÇÃO EXTRA]**
Função que limpa o array de um dado Bloom Filter, criando um novo, vazio.
Inputs: bf (Bloom FIlter criado anteriormente)
Output: obj (novo Bloom FIlter)

O teste deste módulo está no ficheiro *Teste.m*, cujo script cria um Bloom Filter, insere uma palavra aleatória, cria outra palavra aleatória e verifica se esta está contida no Bloom Filter. No entanto **recomendamos o uso da aplicação final (appFinal)** para verificar na íntegra (e melhor) o uso deste módulo.

- **Módulo Finding Similar Items**

Para a execução deste módulo, foi criado um script que chama as funções necessárias.

Script *FindingSimilarItems (FindingSimilarItemsF.m)*
Executa e chama as funções necessárias à execução correcta deste módulo.
Cria um set de shingles para um dado documento, faz o minHash utilizando a técnica ManyHash utilizando uma função hash, e mostra a similaridade de Jaccard para um par de documentos.

Este script é também utilizado como função da appFinal, no ficheiro seguinte:

- *FindingSimilarItems (FindingSimilarItems.m)*
Input: Doc1 (documento para comparação) e Doc2 (documento para comparação)
Output: jaccardSimilarity (Similiaridade de Jaccard entre dois documentos)

As funções utilizadas neste módulo foram as seguintes:

- *doc2set (doc2set.m)* **[NÃO FUNCIONA]**
Função que cria um set de shingles (suportada pela função createShingle) para um dado documento.
(Tentámos fazer esta função funcionar para evitar que o script FindingSimilarItems tivesse um número maior de linhas de código, e após fazer o debug não conseguimos por a função a funcionar. Uma vez que o código está correcto e funciona fora da função, decidimos optar por esta última solução, bem como o facto de que os documentos são comparados em pares, logo apenas precisamos de usar este trecho de código cerca de 2 vezes.)
Inputs: document (documento a ser analisado), shingleSize (tamanho dos shingles, também conhecido na teórica como k)
Outputs: theSet (set de shingles)

- *createShingle (createShingle.m)*
Função que cria um set de shingles, de acordo com um dado set de strings e tamanho para cada shingle. Inclui um exemplo simples em Português que foi utilizado como "dummy" para a criação do código.
Inputs: stringSet (set de strings do documento) e shingleSize (tamanho dos shingles, também conhecido na teórica como k)
Outputs: shingle (set de shingles)

- *removechars (removechars.m)*
Função que retira caracteres de uma dada string.
Inputs: str (string a ser (ou não) alterada) e chars (caracteres a ser retirados da string)
Output: str (string com os caracteres removidos)

- *string2hash (string2hash.m)*
Função hash que gera um valor hash a partir de uma string. A diferença entre esta função e a função utilizada no módulo Bloom Filter é que esta devolve um valor de hash até ao máximo suportado pelo tipo inteiro (2^32-1).
Retirada de http://www.mathworks.com/matlabcentral/fileexchange/27940-string2hash/content/string2hash.m
Inputs: str (string a ser gerada) e maxSize (tamanho máximo da hash a gerar)
Output: hash (valor hash gerado pela função – entre 0 e 2^32-1)

- *hashShingle (hashShingle.m)*
Função que irá percorrer todo o set de shingles e calcular o valor de hash de cada um.
Inputs: shingleSet (set de shingles a ser calculado)
Output: shingleHashed (set de shingles com valor de hash em cada índice)

O teste deste módulo está no ficheiro *FindingSimilarItems.m*, no entanto tal como descrito para o Bloom Filter, recomendamos vivamente o uso da aplicação final (appFinal) para testar este módulo, uma vez que se poderá visualizar melhor e com um exemplo prático o seu

funcionamento. O código não tem alterações comparativamente à função *FindingSimilarItems.m* que descrevemos acima, por isso também apelamos ao uso do *appFinal.m* (e a opção 2 do menu) como teste deste módulo.

- **Trabalho Final - appFinal**

Para a execução do trabalho propriamente dito, foi criado um script que chama as funções dos dois módulos e as combina tendo assim como base a ideia original, ou seja, um script que fosse capaz de mostrar ao utilizador se já tinha visitado aquela página e recomendar-lhe páginas semelhantes.

Script *appFinal (appFinal.m)*
Executa e chama as funções necessárias à execução correcta dos dois módulos.
Cria um BloomFIlter, cria um set de shingles para um dado documento, faz o minHash utilizando a técnica ManyHash utilizando uma função hash, e mostra a similaridade de Jaccard comparando o "site" inserido pelo utilizador com todos os outros e procurando imediatamente similares. Neste caso consideramos "sites" similares os que tenham uma similaridade de Jaccard superior a 0,1 e que não estejam no histórico do utilizador.

Modo de Utilização:

Necessita do histórico do utilizador (se existir), e deve ser colocado no ficheiro "historicoWeb.txt". Precisa também que todas as páginas com "sites" sejam colocadas na pasta do mesmo e seus nomes colocados em "documentos.txt".

O script dá ao utilizador todas as informações necessárias para que este funcione correctamente. Pede que o utilizador insira o tamanho do BloomFilter desejado e também o número de funções de hash pretendido. Está incluído um modo de debug para que seja visualizado o valor das variáveis que a aplicação cria no final. Durante a execução do script algumas variáveis são limpas de forma a poupar memória, pois já não estão a ser utilizadas e não são mais precisas, não afectando o decorrer do script.

O script tem um menu principal com três opções:

- Verificar se já visitou um "site". Para esta função o utilizador deve selecionar **1** no menu principal e depois introduzir o nome do "site" que pretende verificar se já foi visitado ou não.

- Procurar "sites" similares. Para esta função o utilizador deve selecionar **2** no menu principal e depois introduzir o nome do "site" que pretende que o script procure similares.

- Sair. Para esta função o utilizador deve selecionar **0** no menu principal.

Comentários:

Confessamos que tivemos várias dificuldades no início deste trabalho (como os professores poderão ver comparando as submissões semanais), no entanto após várias pesquisas e um estudo mais aprofundado da teoria e dos conceitos fundamentais dos módulos e seu objectivo foi-nos possível trabalhar e efectuar (na nossa perspectiva) um bom trabalho. No conjunto de submissões, este relatório e conjunto de scripts e funções é o que tem mais alterações pois antes desta entrega final ligámos o "turbo" e conseguimos chegar a algo que pudéssemos de facto entregar e ser avaliado, bem como um exemplo prático de como ambos os módulos funcionam, que foi pensado com vista a ser distinto dos nossos colegas (apesar de não sabermos como serão os trabalhos deles). Apesar de tudo, não conseguimos implementar o conceito **Locality-Sensitive Hashing** falado nas aulas (juntamente com o conceito MinHash) mas reconhecemos a sua utilidade e neste caso em concreto seria outra técnica a implementar de forma a agilizar o processo de similaridade entre documentos.

Em suma, podemos afirmar que este trabalho final foi bastante enriquecedor para ambos, ficando a conhecer conceitos que poderão ser úteis no futuro, caso um de nós queira optar pelo conceito Big Data e tratamento de dados, já utilizados por várias empresas (entre elas a Amazon, como o professor António Teixeira referiu nas aulas teóricas) e cuja análise de dados é importante para reconhecer o funcionamento e melhorar lacunas existentes.