Universidade de Aveiro

Métodos Probabílisticos para Engenharia Informática

MIECT

Relatório do Trabalho Prático "Jaccard's Clinic"

Autores:	*Número Mecanográfico:*
Bruno Caseiro	88804
João Tomás Simões	88930

Dezembro 2018

Contents

1 Lista de Ficheiros

- ContadorEstocastico.java
  Classe Java com a implementação do primeiro módulo.
- CountingBloomFilter.java
  Classe Java com a implementação do segundo módulo.
- MinHash.java
  Classe Java com a implementação do terceiro módulo.
- JaccardsClinic.java
  Programa de demonstração com a aplicação dos 3 módulos em conjunto.
- TestContadorEstocastico.java
  Classe Java para testar o módulo Contador Estocástico desenvolvido.
- TestCountingBloomFilter.java
  Classe Java para testar o módulo Counting Bloom Filter desenvolvido.
- TestMinHash.java
  Classe Java para testar o módulo MinHash desenvolvido.
- Sintomas.txt
  Ficheiro de texto com algumas doenças e seus repetivos sintomas.
- Lorem.txt
  Ficheiro de texto com 1000 palavras gerado no site `https://pt.lipsum.com/`.
- Relatorio.pdf
  Documento que está agora a ler que contém explicações e instruções acerca do código desenvolvido.
- Apresentacao.pdf
  Slides para auxiliar a apresentação do trabalho na aula prática.

2 Módulos

2.1 Contador Estocástico

Neste módulo implementámos as duas maneiras de realizar um contador estocástico que nos foram ensinadas nas aulas teóricas: a primeira e a mais simples, em que o contador aumenta (em média) metade das vezes pois a probabilidade de adicionar

um novo elemento é sempre 0.5; e a segunda e um pouco mais complexa, em que o contador é incrementado com uma probabilidade cada vez menor à medida que o seu valor aumenta (através da fórmula $2^{(-n)}$, sendo `n` o número de elementos que o contador já contém).

Para distinguirmos as duas soluções de contadores possíveis utilizámos dois construtores: um vazio, que nos diz que vai ser usada a primeira maneira e um que aceita um qualquer número inteiro e que nos diz que vamos utilizador um contador estocástico pela segunda maneira.

O resto dos métodos implementados são triviais e intuitivos demais para necessitarem de ser explicados.

2.2 Counting Bloom Filter

Neste segundo módulo, implementámos código para instanciar bloom filters. Criámos 3 construtores diferentes: um em que apenas se personaliza o tamanho do bloom filter, usando o programa um número "ideal" de hash functions e a probabilidade de falsos positivos predefinida (0.01); outro em que se define o tamanho do bloom filter mas também a probabilidade de falsos positivos, continuando o programa a utilizar um número "ideal" de hash functions; finalmente, um último em que se indica o tamanho do bloom filter e o número de hash functions pretendido, calculando o programa automaticamente a probabilidade de falsos positivos através da fórmula dada nas aulas teóricas.

Para gerar as chaves, utilizámos uma hash function personalizada por nós (`stringToHashCustom()`) e ainda a hash function `hashCode()` predefinida do Java (`stringToHash()`). Apartir daí, implementámos todos os métodos para modificar o bloom filter, tais como adicionar, remover, contar e verificar elementos para lidar com ambas as hash functions.

2.3 MinHash

O nosso terceiro módulo tem duas estruturas de dados muito importantes na sua base, o ArrayList(ArrayList(String)) setSaver, e o ArrayList (String) shingleSaver. O setSaver é responsável por, em cada posição, guardar os respetivos shingles de cada set. O outro ArrayList, shingleSaver, guarda todos os shingles de cada set.

Temos dois métodos para efetuar o chamado "Shingling". O charShingle e o wordShingle, ambos recebem a String a ser "shingled" como argumento. O número de caracteres ou palavras para ser feito o "Shingling" é referido pelo utilizador no construtor da class MinHash. Depois deste processo é feita a matriz de 1s e 0s. As colunas possuem os sets e as linhas representam os shingles. Se o set contém aquele shingle, é adicionado um 1. Caso contrário, adiciona-se um 0. Este processo é feito automaticamente a partir dos dois ArrayLists, com o método updateMatrix.

Está na altura de compararmos os sets. Para isso, é preciso calcular a distância de Jaccard das assinaturas de cada set. As assinaturas são calculadas com a ajuda do método getSignature, que preserva sempre a distância de Jaccard dos sets quando cria as respetivas assinaturas. Para criar uma assinatura, são usadas k hash functions (referida também no construtor, ou 500 por default) para cada shingle do set em questão. Depois, cada shingle tem um ID único, que introduzido na função uniHash resulta num pequeno código. No fim de todos os shingles terem passado por esta função, apenas guardamos o menor código, ou seja, o mínimo (minimum hash).

De seguida, fazemos o mesmo para cada hash function. Os mínimos são concatenados por ordem e resultam numa assinatura. O tamanho da assinatura será igual ao número de hash functions usadas. Após este processo todo, são comparadas as distâncias de Jaccard das assinaturas, que serão aproximadamente iguais ao valor teórico. Quanto mais funções de hash forem usadas menor será o erro.

3 Programa "Jaccard's Clinic"

3.1 Explicação

Este programa consiste num hipotético paciente que deseja saber que doença tem através dos sintomas que manifesta. Este foi o tema que escolhemos para a nossa demonstração.

Também neste programa ainda mostramos um histograma com a contagem de todas as doenças detetadas pelos sintomas inseridos, para que o utilizador possa analisar e ver quais e quantas doenças foram detetadas através dos sintomas fornecidos.

3.2 Execução

Para executar o programa, basta compilar todos os módulos no terminal usando o comando `javac *.java` e de seguida executar o programa de demonstração através de `java JaccardsClinic`. Depois disso, estará a correr o programa e basta seguir as instruções que lhe são indicadas pelo terminal.

3.3 Bugs

Até agora não foram detetados erros no nosso código. Todos os testes (apronfundaremos melhor na próxima secção deste relatório) deram os devidos valores esperados e as exceções mais importantes foram tratadas de forma a que o programa não termine bruscamente, como por exemplo, quando o programa pede um número e o utilizador digita uma letra.

O nosso código apenas apresenta duas limitação. A primeira é que quando são inseridos muito poucos sintomas e/ou sintomas pouco específicos, o programa não é capaz de detetar a doença e o resultado pode não ser o mais adequado por causa dessa redundância. Daí recomendarmos que o utilizador tem de ser o mais específico possível ao inserir os sintomas.

A segunda limitação é a impossibilidade de gravar dados. O programa não possui uma base de dados, por isso após o programa ser terminado, todos os dados introduzidos anteriormente serão perdidos.

4 Testes

Os testes realizados não dependem da ação de um utilizador de forma propositada para não ocorrerem erros ou manipulação de resultados. O programa é executado automaticamente e apenas são mostrados no terminal os resultados finais. Em cada ficheiro de testes estão escritos em comentário os respetivos resultados esperados. Os resultados impressos pelo terminal devem ser iguais a esses resultados esperados em comentário. No caso de dependerem de probabilidades, esses valores estarão devidamente identificados (também em comentário) e, por isso, os resultados não deverão ser exatamente iguais mas sim muito parecidos aos esperados.

5 Contribuição dos Autores

Este trabalho resultou da interação e colaboração de ambos os membros do grupo, quer a nível de escrita de código, quer a nível criativo. A troca de ideias entre todos os membros permitiu a elaboração de um trabalho mais completo e de maior qualidade, pelo que todos os membros foram igualmente importantes e contribuiram para a realização deste projeto.

Todas as pessoas deste grupo (Bruno Caseiro e João Tomás Simões) trabalharam de igual forma para a realização deste projeto, tendo cada um realizado aproximadamente a mesma percentagem de trabalho (50%).

Segue agora uma captura de ecrã das estatísticas do repositório geradas pelo GitHub, para uma noção mais detalhada sobre o trabalho que cada um desenvolveu:

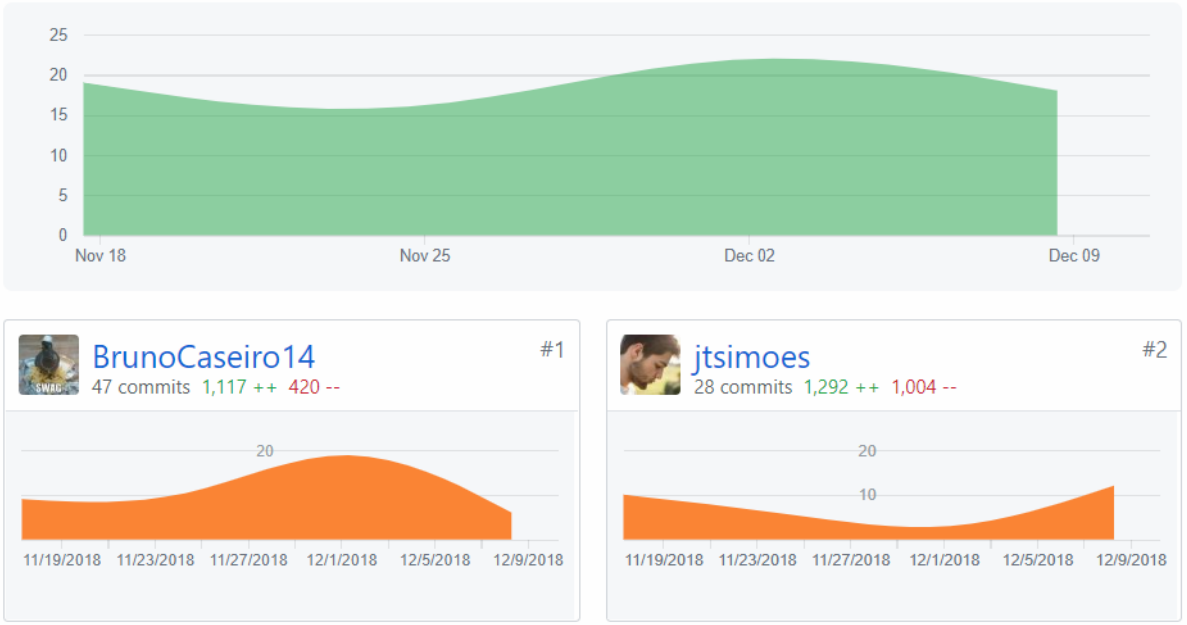

Figure 1: Estatísticas do repositório deste trabalho