universidade de aveiro

DETI | ua

2º Relatório

Recuperação de Informação

2018-2019 1ºS

11-11-2018

Professor: Sérgio Matos

Aluno: André Cardoso, 65069, (marquescardoso@ua.pt)
Ivo Angélico, 41351, (ivoangelico@ua.pt)

Índice:

- Recuperação de Informação
  - Índice:
    - Introdução
    - Utilização
    - Corpus Reader
    - Simple Tokenizer
    - Improved Tokenizer
    - Indexer
    - Indexer merging
    - Mecanismos de Controlo
    - Resultados
    - Conclusões
  - Anexo I - Diagrama de Classes

Introdução

Este relatório é uma continuação da proposta de trabalho inicial que consistiu no desenvolvimento de vários módulos com o objectivo de fazer pesquisas úteis ou retirar informação menos perceptível de um outro conjunto de informação como é o exemplo neste contexto fazer pesquisa eficiente sobre reviews dos clientes da Amazon Customer Reviews Dataset disponível aqui.

O trabalho anterior foi utilizado como base tendo sido alterado o indexer de forma a se considerar pesos no lugar da frequência de termos. Para além disso foram desenvolvidos os módulos necessários para ranking de pesquisa. O ranking teve por base a estratégia lnc.ltc.

De forma a completar a totalidade do trabalho, serão feitos desenvolvimentos adicionais para criar um index posicional que permita pesquisas de proximidade.

Os módulos que tinham sido desenvolvidos no trabalho anterior como *corpus reader, tokenizer, indexer*, controladores para manipular segmentos das estruturas em memória e disco, mecanismos para gestão eficiente da mesma e escrita e leitura de ficheiros de grande dimensões através de sistemas de buffer e cache foram re-organizados para uma melhor estrutura lógica. Neste momento existem os seguintes módulos: models (que incluem as classes que representam as estruturas bases tal como os Postings ou o dicionário), operations.io, operations.segments, core, core.tokenizer e management.

Para seguir a estratégia lnc, o indexer vai primeiro calcular a term frequency e usar esse valor para calcular o peso do termo no documento.

$$\mathrm{wf}_{t,d} = \begin{cases} 1 + \log \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Neste caso não se irá combinar este peso com a frequência do documento invertida ou seja irá considerar-se idf = 1. Por fim é necessário normalizar os pesos multiplicando cada um dos pesos anteriores por:

$$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$$

As queries também serão tratadas mas com a estratéfia ltc. Depois de passarem pelos mesmos processos de normalização que os termos, será calculado um term frequency na query. A partir dessa frequência será calculado um peso à semelhança do que foi feito para os termos. Neste caso já sabemos quantos documentos estão disponíveis e em quantos documentos aparece o termo, por isso irá combinar-se esse peso com a frequência do documento invertida.

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

com

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}$$

Após isto, o peso irá ser normalizado da mesma forma como foram os termos do indexer. O cálculo final do score da query é feito através da seguinte fórmula:

$$Scores[d] += w_{t,d} \times w_{t,q}$$

Utilização

Para gerar ficheiro de execução .jar a partir do código fonte, necessitamos de estar dentro da pasta do projecto e executar o comando.

```
mvn package
```

Para executar podemos fazer uso do comando -h para nos ajudar na orientação na execução das funcionalidades da aplicação. Obtemos a seguintes funcionalidades:

```
java -jar Assignment02.jar -h
```

```
C:\Users\aCard0s0\Desktop\Assignment02\target>java -jar Assigment02-1.0-SNAPSHOT-jar-with-dependencies.jar -h
usage: java -jar name.jar [-cr <arg>] [-h] [-it] [-m <arg>] [-v]
Assignment from Information retrivel class at Aveiro's University
 -cr <arg>       Source .tsv file for doc creation
 -h              Print usage menu
 -it             It use an Improved Tokenizer to generate the terms and
                 Stopword.
 -m <arg>        Sets the max memory to be use.
 -v,--version    Print the version of the application

Developer by: AndrÚ Cardoso 65069 & Ivo AngÚlico 41351
```

Figura 1. Opção de manipulação do ficheiro de execução.

A execução por definição da aplicação, sem qualquer tipo de argumento de entrada, tem o seguinte comportamento:

1. Usa como leitura um ficheiro … que tem de estar contido no mesmo directório que ficheiro de execução;
2. Usa 1024 de memória RAM;
3. Passa os tokens pelo Simple Tokenizer

Corpus Reader

O Corpus Reader utiliza um BufferedReader para ler o ficheiro, lendo um documento de cada vez. O Corpus Reader retorna um objecto, Doc que contém um *document identifier (id)* que é um inteiro único atribuído sequencialmente a cada objecto. O objecto Doc tem ainda o conteúdo do documento, já expurgado das secções irrelevantes. Se neste trabalho fosse necessário executar queries o Doc deveria ter guardadas informações adicionais que seriam necessárias para a resposta às queries tais como *customer_id*, *review_id* e *product_id*.

Após se ler um documento este será processado por um tokenizer. Dependendo dos argumentos de entrada o tokenizer será um dos dois seguintes.

Simple Tokenizer

Este tokenizer separa o documento em cada espaço, transformar os caracteres alfabéticos para lowercase, remover os caracteres não alfabéticos e remover os tokens com menos que 3 caracteres.

Improved Tokenizer

Este tokenizer separa o documento em cada espaço e transformar os caracteres alfabéticos para lower case.
Após isso verifica uma série de regras:

1. Verifica se o token tem um arroba @. Se tiver verifica se tem pelo menos um caractere à esquerda, pelo menos 3 caracteres à direita e se um desses 3 caracteres. Se essa condição for preenchida assume-se que este termo é um email e por isso deve ser mantido.
2. Verifica se o termo tem uma apóstrofe ‘ e se tiver divide o termo em duas partes. Se a segunda parte tiver até 2 caracteres assume-se que se trata de uma contracção e por isso é removida ficando apenas a primeira parte do termo. Os restantes caracteres não alfabéticos são removidos. Se não, a apóstrofe é removida bem como os restantes caracteres não alfabéticos.
3. Verifica se o termo tem um hífen - e se tiver divide o termo em duas partes. Se a primeira parte tiver até 2 caracteres assume-se que se trata de um prefixo e por isso a primeira parte é removida. Os caracteres não alfabéticos são removidos.
4. Passam pelo Porter Stemmer e são depois adicionados a uma lista de termos

Após estas regras é verificado se o token faz parte de uma lista de stopwords. Esta lista foi constituída a partir da lista presente aqui, à qual foram aplicadas as regras anteriores de forma aos termos que tinham sido normalizados serem devidamente identificados.

Indexer

O indexer foi implementado segundo a estratégia *Blocked Sort-Based Indexing* pois o dicionário é relativamente pequeno e cabe em memória mas as *posting lists* ocupam demasiado espaço sendo necessário escrever em disco. Nesta secção iremos analisar a construção dos sub-segmentos do indexer que estarão ordenados e na secção iremos analisar a junção dos segmentos num index final.

O indexer recebe um docId e uma lista de termos. Os termos são corridos e é calculada o seu peso com base no processo descrito anteriormente. A seguir é verificado se o termo já existe na hashmap do indexer onde estão os termos e posting list. Se não existir é criada uma *posting list* nova e adicionada ao hashmap. Se existir, a *posting list* existente é alterada para acrescentar este documento e respectivo peso.

Após isto é verificado quanta memória está a ser utilizada. Se o valor for superior ao definido como máximo aceitável o hashmap é convertido num **treeMap** de forma aos termos ficarem ordenados e é salvo para o disco com recurso a um newBufferedWriter. Foi estudada a possibilidade de ser criado de origem um treeMap em vez do hashmap mas foi verificado que nesse caso a operação de indexação é aproximadamente 30% mais lenta do que quando era criado um hashmap e depois convertido para treeMap. Após se escrever em disco, o hashmap é limpo de forma a se começar uma nova lista. O garbage collector é chamado para se limpar a memória.

Indexer merging

O processo anterior é executado continuamente até não haver mais documentos para indexar. Aí é escrito para memória o último hashmap. Temos agora em disco uma série de documentos com segmentos do indexer final cujos membros necessitam de ser reunidos em ordem alfabética. O indexer final será composto com base num LinkedHashMap.

O primeiro termo de cada segmento é lido. Após isso entra-se num ciclo até todos os termos de todos os segmentos terem sido indexados. Para decidir qual o próximo termo a ser inserido no dicionário, são comparados um elemento de cada segmento até ser escolhido qual o de maior ordem lexicográfica. Esse elemento é adicionado ao LinkedHashMap ou a posting list é fundida com a já existente se o termo já tiver sido adicionado. É lido um novo elemento do segmento (ou segmentos caso o mesmo estivesse disponível em vários segmentos) de onde o termo foi lido (caso ainda haja termos para serem indexados nesse segmento). É realizado o mesmo processo de salvar em disco o LinkedHashMap quando a memória disponível é baixa. Em memória é mantido um dicionário com o termo, um id do documento com o segmento do index final onde está guardado esse termo e a linha em que ele está.

Mecanismos de Controlo

Durante o desenvolvimento da solução mecanismos de leitura e de escrita tanto de documentos (linha) como de segmentos (descargas feitas para disco da informação das estruturas que chegaram ao limite da memória), tendo em consideração boas práticas de programação.

A instanciação objecto Memory com um argumento Inteiro que representa a quantidade de memória disponível para a JVM e outro Double que define a percentagem limite de uso para iniciar os mecanismo de libertação de memória. O Timer é utilizado para coletar informações sobre o desempenho temporal da aplicação.

Nas classes responsáveis pela manipulação de cheiros estão implementadas com mecanismos de buffering, assim como mecanismos de stream para concatenação de Strings em memória.

Depois das operações de indexação, é reservado 60% da memória para manter os segmentos. Quando é feita pesquisa de termos é verificado se o segmento já se encontra já carregado para um pesquisa mais eficiente. Caso não esteja, e a memória chegou ao limite, é descarregado o segmento menos usado nas pesquisas anteriores.

Ranked Retrieval

Este módulo é responsável por aplicar os filtros aos termos da query semelhantes aos aplicados aos termos do indexer e por verificar se os mesmos se encontram no dicionário. Se a query estiver presente no dicionário, as linhas adequadas dos respectivos segmentos do indexer são lidas. Com os postings lidos e com base no processo anteriormente descrito é calculado um score para a query.

Resultados

Query

Depois da indexação feita podemos verificar usamos o termos Galaxy como pesquisa e foi-nos devolvido o seguinte documento com mais relevância:

US 49906433 R1H4ZTIYYT6KMJ B007Z0WEH0 333315966 SAMRICK - PURPLE - High Capacitive Aluminium Stylus Pen for Samsung S5300 Galaxy Pocket - i9220 Galaxy Note - i8160 Galaxy Ace 2 - S6500 Galaxy Mini 2 - i9070 Galaxy S Advance - i9100 Galaxy S 2 - i9300 Galaxy S 3 - S5360 Galaxy Y - i8150 Galaxy W - S5830 Galaxy Ace - S5570 Galaxy Mini - i9000 Galaxy S - i9250 Galaxy Nexus - S7500 Galaxy Ace Plus - i9001 Galaxy S Plus - S5670 Galaxy Fit - S5660 Ga Wireless 5 0 0 N Y Purple Stylus My favorite color is purple. It works great. I use it on my ipod and touch screen cell phone. Love it.

Anexo I - Diagrama de Classes