Universidade do Minho
Escola de Engenharia

Bibliotecas Digitais

TP1

Alunos:

João Miguel da Silva Alves (83624)

Paulo Jorge Alves (84480)

Docente:

Joaquim Macedo

MESTRADO INTEGRADO EM ENGENHARIA BIOMÉDICA
INFORMÁTICA MÉDICA 2020/2021

O Terrier é uma plataforma modular para o rápido desenvolvimento de aplicações de recuperação de informação em grande escala. É uma ferramenta altamente flexível, que implementa o estado da arte da indexação e das funcionalidades de recuperação, além de fornecer uma ótima plataforma para desenvolvimento, avaliação e experimentação a respeito da recuperação de textos.

I. PARTE TEÓRICA

1) Relativamente à coleção do Terrier.

a. Em que ficheiros se coloca os parâmetros que determinam o modo de funcionamento do Terrier?

O Terrier é configurado em geral por alguns ficheiros, todos no *etc/directory*.

Os ficheiros que mais focam nestas configurações são o *terrier.properties* (exemplo na figura abaixo) e o terrier-log.xml.

No arquivo *terrier.properties*, as propriedades são especificadas no formato nome = valor (formato padrão das propriedades Java). Os comentários são linhas que começam com #.

```
#document tags specification
#for processing the contents of
#the documents, ignoring DOCHDR
TrecDocTags.doctag=DOC
TrecDocTags.idtag=DOCNO
TrecDocTags.skip=DOCHDR

#query tags specification
TrecQueryTags.doctag=TOP
TrecQueryTags.idtag=NUM
TrecQueryTags.process=TOP,NUM,TITLE
TrecQueryTags.skip=DESC,NARR

#stop-words file
stopwords.filename=stopword-list.txt

#the processing stages a term goes through
termpipelines=Stopwords,PorterStemmer
```

b. Diga como se indica como se usa ou não a expansão de interrogações, a lista de stop-words e o stemming?

O Terrier fornece a funcionalidade de "expansão de *queries*". Usando a configuração a seguir, extraem-se os 40 documentos mais informativos sobre a *query* pesquisada.

Os termos extraídos dos 10 principais documentos é que serão os termos recuperados.

As configurações poderão ser as seguintes:

```
expansion.documents=3
expansion.terms=10
```

Se as duas propriedades acima não forem especificadas, no ficheiro *terrier.properties*, o Terrier usará como predefinição: #expansion.documents = 3 e #expansion.terms = 10.

O Terrier aplica um mecanismo de expansão de consulta sem parâmetros, por predefinição. Uma abordagem alternativa é a utilização do algoritmo de Rocchio.

Para se poder utilizar este algoritmo, o usuário deve definir a propriedade #parameter.free.expansion como falsa, no ficheiro terrier.properties.

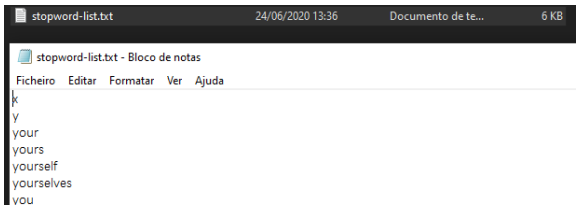
```
#parameter.free.expansion=false
```

Neste mesmo ficheiro, para definir o parâmetro beta do algoritmo de Rocchio é necessário escrever: rocchio.beta = X, sendo X um número. Por exemplo: rocchio.beta=0.8.

Se o parâmetro rocchio.beta não for especificado no ficheiro terrier.properties, o Terrier aplica este valor como 0.4, por default.

O ficheiro de stopwords, denominado de “stopword-list.txt” encontra-se na pasta “Share”.

```
stopwords.filename=stopword-list.txt
```



No ficheiro terrier.properties, encontra-se a seguinte setting:

```
termpipelines=Stopwords,PorterStemmer
```

Usando a configuração padrão acima, o sistema interrompe um termo do stopwords.txt e, em seguida, aplica o algoritmo Stemming - processo de reduzir palavras flexionadas/derivadas à sua raiz - se não for uma stopword.

Se for para manter todas as stopwords e aplicar uma versão fraca do Porter's stemmer, a configuração deve ser a seguinte: `#termpipelines=WeakPorterStemmer`

Os stemmers mudaram desde o Terrier 2.0. Se se pretender usar um stemmer mais antigo, deve-se usar: TRv2PorterStemmer e TRv2WeakPorterStemmer.

Se o objetivo for manter todas os termos inalterados, usa-se: `#termpipelines=`

c. Em termos do processamento das interrogações e dos documentos que variantes estão disponíveis?

Em termos do processamento das interrogações e dos documentos, as variantes disponíveis são: processar documentos retirando as stopwords, o uso do stemming (lematização) nas palavras da query, e a expansão das interrogações. Esta expansão significa que para além dos termos que os utilizadores colocam, colocam-se mais termos nas interrogações que se vão buscar aos documentos. Faz-se a busca no "search engine", pega-se nos documentos daqueles termos escolhidos e vê-se quais os que ocorrem mais nesses mesmos documentos.

Por fim, acrescentar esses termos à interrogação, fazendo de novo a interrogação.

d. Indique outras possibilidades de configuração que não foram explicitadas, embora estejam disponíveis.

Outra possibilidade de configuração relaciona-se com a construção do index.

Este pode conter só as palavras ou, além destas, conter a informação de posição de cada uma nos vários documentos, ficando desta forma mais extenso.

```
#indexing settings:
################################################################

#collection.spec is where a Collection object expects to find information
#about how it should be initialised. Eg for a TREC collection, its a file
#containing the list of files it should index.
collection.spec=/local/collections/WT2G/files.txt
#default collection.spec is etc/collection.spec

#To record term positions (blocks) in the index, set
#block.indexing=true

#block indexing can be controlled using several properties:
#blocks.size=1 = this property records the accuracy of one block
#blocks.max - how many blocks to record in a document

#fields: Terrier can save the frequency of term occurrences in various tags
#to specify fields to not during indexing
FieldTags.process=TITLE,ELSE
#note that ELSE is a special field, which is everything not in one of the other fields
```

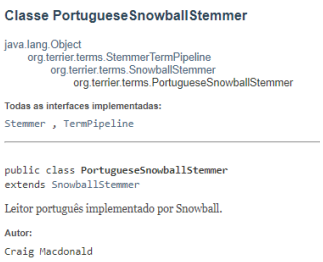
2) Fixemos agora a atenção na configuração da indexação no Terrier.

a. O primeiro aspeto a parametrizar é o componente TermPipeline. Existe algum stemmer para português no Terrier? E lista de stopwords?

O Terrier tem a capacidade de registar a frequência com que os termos ocorrem em vários conjuntos de documentos.

O indexador itera através dos documentos da coleção e envia cada termo encontrado por meio do TermPipeline. Este transforma os termos e pode remover termos que não deveriam ser indexados. As configurações desta componente já estão explicadas na questão 1b).

Sim, existe um stemmer para Português no Terrier, denominado de "PortugueseSnowballStemmer".



Classe PortugueseSnowballStemmer

java.lang.Object
  org.terrier.terms.StemmerTermPipeline
    org.terrier.terms.SnowballStemmer
      org.terrier.terms.PortugueseSnowballStemmer

Todas as interfaces implementadas:

`Stemmer , TermPipeline`

```
public class PortugueseSnowballStemmer
extends SnowballStemmer
```

Leitor português implementado por Snowball.

Autor:

`Craig Macdonald`

Sim, existe uma lista de StopWords em Português, lista esta que se encontra no ficheiro "stopwords_basedTextTilling.txt" – ficheiro usado na resolução do exercício prático abaixo.

b. Para manter a meta-informação dos documentos existe o meta-index. O que se pode configurar aí?

Os metadados do documento são registados num meta-índice. Por exemplo, esses meta-dados podem incluir o DOCNO e a URL de cada documento, que o sistema pode usar para representar o documento durante a recuperação. O meta-índice pode ser configurado para tomar nota de vários atributos do documento durante a indexação. O nome das entradas no meta-índice deve ser igual aos nomes das tags. Os atributos disponíveis dependem daqueles fornecidos pela implementação do documento. O meta-índice pode ser configurado usando as seguintes propriedades:

```
#Next we need to tell the meta index to save all of these extra document properties
#so that they are not thrown away. In addition to the docno which is always available,
#and the title and url tasks saved by TaggedDocument, we will also save the url and
#crawldate which are added to each document by the TRECWebCollection class.
#indexer.meta.forward.keys=docno,title,body,url,crawldate

#Again, we need to specify how many characters long each meta index entry is.
#Note that the title and body
#abstract lengths should be the same here as in TaggedDocument.abstracts.
#indexer.meta.forward.keylens=32,120,500,140,35
```

- indexer.meta.forward.keys - Lista delimitada por vírgulas de atributos do documento para armazenar no meta-índice. Por exemplo: indexer.meta.forward.keys = docno ou indexer.meta.forward.keys = url, title. Se esta propriedade for definida, a propriedade a seguir também precisa ser definida.
- indexer.meta.forward.keylens - Lista delimitada por vírgulas do comprimento máximo dos atributos a serem armazenados no meta-índice. O valor default é 20. O número de comprimentos de chave aqui deve ser idêntico às chaves numéricas em indexer.meta.forward.keys.

- indexer.meta.reverse.keys - Lista delimitada por vírgulas de atributos de documento que denotam exclusivamente um documento. Isso significa que, dado um valor de atributo de documento, um único documento pode ser identificado.

c. Relativamente ao processo de indexação existem duas alternativas: indexação num único passo e indexação em dois passos. Quais são as diferenças fundamentais? Apresente também vantagens e desvantagens.

O Terrier oferece dois tipos de indexação: indexação num único passo e indexação em dois passos.

Ambos os métodos criam um índice invertido idêntico, que produz uma eficácia de recuperação idêntica. No entanto, diferem noutras características, nomeadamente no seu suporte para expansão de query e na escalabilidade e eficiência ao indexar grandes quantidades de documentos. A escolha do método de indexação depende da necessidade de expansão da query e pela quantidade de dados com os quais se trabalha. Em particular, uma diferença entre os dois métodos é que, apenas na indexação em dois passos se cria diretamente um índice direto, que é usado para a expansão da query, sendo por isso uma vantagem. No entanto, este tipo de indexação não é dimensionado para grandes quantidades de documentos (o máximo é cerca de 25 milhões de documentos). Uma vantagem da indexação num único passo é ser mais rápida, mas não cria um índice direto, sendo esta uma desvantagem deste método em relação ao anterior. Neste método de indexação num único passo, em vez de se construir um arquivo direto da coleção, as posting lists dos termos são mantidas em memória e gravadas no disco quando a memória se esgota. Se eventualmente não for criado um índice que não tenha um índice direto, é possível criá-lo mais tarde usando o comando “inverted2direct” do Terrier.

Na Bash, para correr a indexação em 2 passos apenas precisamos de escrever “bin/terrier/batchindexing”, enquanto que para correr a indexação em apenas 1 passo tem-se: “bin/terrier/batchindexing -j”.

d. O que é o BlockIndexer? Podemos indexar com informação de posição? Como?

Um BlockIndexer é um índice que permite guardar informação de blocos para indexação de termos num documento.

Um bloco é uma unidade de texto num documento. A indexação por blocos permite ao Terrier guardar as informações de posição de cada termo. Dependendo de como o Terrier for configurado, um bloco pode ser de tamanho 1 ou superior. Tamanho 1 significa que a posição exata de cada termo pode ser determinada. Para tamanhos superiores a 1, o id do bloco é incrementado após cada N termos. É possível configurar o tamanho de cada bloco usando a propriedade `blocks.size`.

Sim, é possível indexar com informação de posição. Ao efetuar o processo de recuperação pode-se habilitar a indexação de bloco definindo a propriedade block.indexing para True, no ficheiro terrier.properties.

```
#To record term positions (blocks) in the index, set
#block.indexing=true

#block indexing can be controlled using several properties:
#blocks.size=1 = this property records the accuracy of one block
#blocks.max - how many blocks to record in a document
```

Quando se ativa a propriedade block.indexing, o TrecTerrier usará o BlockIndexer e não o BasicIndexer.

Ao carregar um índice, o Terrier detetará que o índice possui informações do bloco guardadas e usará as classes apropriadas para ler os arquivos do índice.

As estruturas de dados de índice criadas irão conter as posições para cada posting list e podem ser lidas e acedidas por meio do comando PostingIndex.getPostings() que irá implementar o BlockPosting para além do IterablePosting.

e. Crédito extra: Para indexação de coleções muito grandes está disponível o Hadoop Map-Reduce Indexing. Explique sucintamente.

Hadoop MapReduce é uma estrutura de software para escrever facilmente aplicações que processam grandes quantidades de dados (conjuntos de dados de vários terabytes) em paralelo com grandes clusters (milhares de nós) de hardware comum de uma forma confiável e tolerante a falhas.

O MapReduce geralmente divide o conjunto de dados de entrada em blocos independentes que são processados pelas “map tasks” de uma maneira completamente paralela. A estrutura classifica as saídas dos mapas, que são então inseridas nas “reduce tasks”. Normalmente, a entrada e a saída do algoritmo são armazenadas num sistema de arquivos. O framework cuida do agendamento de tarefas, monitorando-as e reexecuta as tarefas com falha.

Normalmente, os nós de computação e os nós de armazenamento são iguais, ou seja, a estrutura MapReduce e o Hadoop Distributed File System estão em execução no mesmo conjunto de nós. Essa configuração permite que a estrutura agende com eficácia as tarefas nos nós onde os dados estão presentes, resultando numa largura de banda agregada muito alta em todo o cluster.

Embora a estrutura do Hadoop seja implementada em Java TM, os aplicativos MapReduce não precisam ser escritos em Java.

Desta forma, o Hadoop (HDFS) armazena dados em ficheiros e não os indexa. Para encontrar algo, é necessário executar o MapReduce, examinando todos os dados. O Hadoop é eficiente onde os dados são muito grandes para uma base de dados. Com conjuntos de dados muito grandes, o custo de regenerar os índices é tão alto que não pode indexar facilmente os dados alterados.

Supondo que existem 2 ficheiros para armazenar no HDFS para processamento. O primeiro tem 500 MB e o segundo tem cerca de 250 MB. Portanto, teremos 4 InputSplits de cerca de 128 MB cada no primeiro ficheiro e 2 InputSplits cada no segundo ficheiro.

Pode-se aplicar 2 tipos de indexação para o caso mencionado:

1. Com a indexação baseada em ficheiros, onde se terminará com 2 ficheiros, o que significa que a consulta indexada será equivalente a uma consulta de scan completa.

2. Com Indexação baseada em InputSplit, de onde resultarão 4 InputSplits. O desempenho deve ser muito melhor do que fazer uma consulta de scan completa.

Para se implementar o índice InputSplits, é necessário realizar as seguintes etapas:

1. Construir um índice a partir de um conjunto de dados – através de um MapReduce para extrair o valor que se pretende indexar e gerá-lo junto com seu InputSplit MD5 hash.

2. Obter o(s) InputSplit(s) para o valor indexado que se está a procurar - a saída do MapReduce será “Reduced Files” (contendo Índices baseados em InputSplits) que serão armazenados em HDFS.

3. Executar o MapReduce apenas nos InputSplits indexados que poderá ser feito pelo Hadoop, uma vez que é capaz de recuperar o número de InputSplit a ser usado, utilizando FileInputFormat.class. De seguida, criar uma classe IndexFileInputFormat estendendo o FileInputFormat.class default e substituindo o método getSplits(). Além disto, é necessário ler o ficheiro criado na etapa anterior, adicionar todos os InputSplits indexados numa lista e depois, comparar essa lista com aquela retornada pela superclasse. Por fim, será retornado ao JobTracker apenas os InputSplits que foram encontrados no seu índice.

4. Na classe Driver, tem-se agora que usar esta classe IndexFileInputFormat. É preciso definir como InputFormatClass fornecendo job.setInputFormatClass (IndexFileInputFormat.class);

3) Vejamos como se configura a recuperação de informação.

a. No modo batch temos que indicar onde o sistema deve ir buscar as interrogações. Explique como.

Após terminado o processo de indexação, pode-se prosseguir com a recuperação da coleção de documentos. Nesta etapa, as propriedades de configuração para aplicação stemming, remoção de palavras irrelevantes ou não, e o comprimento máximo dos termos, devem ser exatamente as mesmas propriedades utilizadas para indexar a coleção.

Em primeiro lugar, na propriedade “trec.topics”, é necessário especificar os ficheiros que contêm as “queries” a serem processadas.

Existem dois formatos de ficheiros de tópicos TREC que são suportados pelo Terrier. No primeiro, os tópicos são marcados em tags semelhantes a XML; o segundo é um formato de texto simples com um tópico por linha, sendo o número do tópico a primeira palavra.

Depois disto, é necessário correr “bin/terrier batchretrieve” na linha de comandos.

b. Como se indica que parte do ficheiro do tópico usar para construir a interrogação?

As tags dos ficheiros de tópicos a serem processados devem ser especificadas antes de processar os arquivos de tópicos XML.

Podemos fazer isto definindo as propriedades TrecQueryTags.process, que denota quais tags processar, TrecQueryTags.idtag, que significa a tag que contém o identificador da query, e TrecQueryTags.skip, que denota que tags de query devem ser ignoradas.

Por exemplo, suponha que o formato dos tópicos seja o seguinte:

```
<TOP>
<NUM>123<NUM>
<TITLE>title
<DESC>description
<NARR>narrative
</TOP>
```

Se o objetivo for ignorar a "description" e a "narrative" (tags DESC e NARR respetivamente) e, consequentemente, usar apenas o título, é preciso configurar:

```
TrecQueryTags.doctag=TOP
TrecQueryTags.process=TOP,NUM,TITLE
TrecQueryTags.idtag=NUM
TrecQueryTags.skip=DESC,NARR
```

Se, alternativamente, o objetivo for usar o título, a descrição e as tags narrativas para criar a query, é preciso configurar as propriedades da seguinte forma:

```
TrecQueryTags.doctag=TOP
TrecQueryTags.process=TOP,NUM,DESC,NARR,TITLE
TrecQueryTags.idtag=NUM
TrecQueryTags.skip=
```

As tags especificadas por TrecQueryTags não fazem distinção entre maiúsculas e minúsculas. Se se quiser que ocorra essa distinção é necessário definir "TrecQueryTags.casesensitive = false".

Os ficheiros de tópicos de linha única têm um formato mais simples, sem a descrição adicional e informações narrativas. O suporte para ficheiros de tópico de linha única é fornecido pela classe SingleLineTRECQuery. Para usar um arquivo de tópicos neste formato, deve-se primeiro definir trec.topics.parser = SingleLineTRECQuery. O comando `batchretrieval` apresenta uma opção -s na linha de comando com o mesmo efeito: bin/terrier batchretrieve -s.

c. Apresente quatro dos métodos usados para ponderação dos termos. Inclua pelo menos um método da família DFR. Explique sucintamente os métodos selecionados.

A divergência da aleatoriedade (DFR) é baseada na seguinte ideia: "Quanto mais a divergência entre a frequência do termo dentro do documento e sua frequência dentro da coleção, mais a informação carregada pela palavra t no documento d. Noutras palavras, o "peso" está inversamente relacionado com a probabilidade de frequência do termo dentro do documento d obtido por um modelo M de aleatoriedade.

$$\boxed{\text{weight}(t|d) = k\text{Prob}_M(t \in d|\text{Collection})}$$

M representa o tipo de modelo de aleatoriedade que utiliza para calcular a probabilidade.

d é o número total de palavras nos documentos.

t é o número de uma palavra específica em d.

k é definido por M.

É possível que se utilizem diferentes modelos para escolher o modelo M apropriado de aleatoriedade. O Terrier fornece implementações de muitos modelos de ponderação. Em particular, alguns dos modelos de ponderação implementados incluem muitos dos que estão associados ao DFR, entre outros:

- BM25:

O BM25, também conhecido como Okapi BM25, ordena um conjunto de documentos baseados nos termos da consulta que aparecem em cada um dos documentos indexados sem levar em consideração a relação entre dois termos dentro do documento, por exemplo seu sentido diferenciado caso seja um substantivo composto.

Esta função é baseada em modelos probabilísticos de recuperação de informação, especificamente o BIR (Binary Independent Retrieval) desenvolvido por Stephen E. Robertson e Karen Spärck Jones nos anos 70.

O BM25 é baseado no conceito de um conjunto de palavras através do qual os documentos que se pretende ordenar são representados com base na sua relevância para uma determinada query.

Dada uma consulta Q, que contém as palavras-chave q1, ..., qn, o valor de relevância atribuído pela função BM25 para o documento D será:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

onde f(qi, D) é a frequência de aparecimento no documento D dos termos que aparecem na consulta Q, | D | é o comprimento do documento D (em número de palavras), e avgdl é o comprimento médio do documentos na coleção que se está a pesquisar. k1 e b são parâmetros que permitem ajustar a função às características específicas da coleção com a qual se está a trabalhar. Embora esses parâmetros geralmente dependam das características específicas de cada coleção, os valores k1 = 2.0 ou k1 = 1.2 e b = 0.75 são valores que foram estabelecidos a partir de experiências realizadas ao longo dos anos nas conferências TREC. IDF (qi) é o peso IDF (frequência inversa do documento) das palavras-chave que aparecem na query Q. Normalmente, o IDF é calculado usando a seguinte função:

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

onde N é o número total de documentos na coleção, e n(qi) é o número de documentos que contêm a palavra-chave qi.

- DLH(DFR): O modelo DFR hiper-geométrico DLH.

DLH é um modelo de divergência de aleatoriedade (DFR) livre de parâmetros que normalmente é implementado como um modelo de ponderação autónomo para aplicações de recuperação. Assume uma distribuição de frequência de termo hipergeométrica que é reduzida a uma distribuição binomial baseada numa distribuição anterior de termo não uniforme. É uma distribuição discreta utilizada para encontrar a probabilidade de retirar X elementos do tipo A numa determinada sequência de n extrações de uma amostra de tamanho N, com M elementos do tipo A e N-M elementos do tipo B, sem que haja reposição das amostras. Definindo o X como uma variável aleatória, essa variável é responsável por encontrar a quantidade de elementos do tipo A. Assim, define-se a distribuição de probabilidade de X como:

$$\mathbb{P}(X = x) = \frac{\binom{M}{x}\binom{N-M}{n-x}}{\binom{N}{n}}$$

Através desses conceitos, o algoritmo implementado no Terrier é definido como:

$$q \cdot \left[\frac{f \cdot \log_2\left(\left[f \cdot \frac{ADL}{docLength}\right] \cdot \left[\frac{n}{ct}\right]\right) + [(docLength - f) \cdot \log_2(1-tf)] + [0{,}5 \cdot \log_2(2 \cdot \pi \cdot f \cdot (1-tf))]}{f-k} \right]$$

onde ADL é a média do comprimento dos documentos, $docLength$ é o comprimento dos documentos, n é a quantidade de documentos indexados, ct é a frequência do termo na coleção de documentos, tf é a razão entre a quantidade de termos dentro de um documento, f é o comprimento do documento, e k é uma constante de normalização.

Conceito de Normalização – explicação teórica como base para os modelos seguintes

Quando um termo raro específico não pode ser encontrado num documento, nesse documento o termo tem probabilidade nula de ser informativo. Por outro lado, se um termo raro ocorre com frequência num documento, ele pode ter uma probabilidade muito alta, próxima de 100%, ou seja, é considerado informativo para o tópico mencionado pelo documento.

Se a frequência do termo no documento for relativamente alta, então, inversamente, o risco do termo não ser informativo é relativamente pequeno. Quanto mais o termo ocorre no conjunto de elite, menos frequência do termo é devido à aleatoriedade e, portanto, menor é o risco associado.

Antes de usar a frequência dentro do documento tf de um termo, o comprimento do documento dl é normalizado para um comprimento padrão sl. Desta forma, as frequências de termo tf são recalculadas em relação ao comprimento de documento padrão, ou seja:

$tf_n = tf * log(1+ sl/dl)$ (normalization 1) , onde tfn representa a frequência de termo normalizada.

Outra versão da fórmula de normalização é a seguinte:

$$tf_n = tf * log(1 + c*(sl/dl)) \text{ (normalization 2)}$$

A normalização 2 é geralmente considerada mais flexível, uma vez que não existe um valor fixo para c. Neste caso, tf é o termo-frequência do termo t no documento d, dl é o comprimento do documento, e sl é o comprimento padrão.

- PL2 (DFR): Estimativa de Poisson para aleatoriedade, sucessão de Laplace para primeira normalização e Normalização 2 para normalização de frequência de termo.

O algoritmo PL2 faz utilização de dois modelos matemáticos combinando-os para sua implementação, a distribuição de Poisson e a normalização euclidiana.

A distribuição de Poisson calcula a probabilidade de que um evento ocorra num determinado intervalo de tempo. Na distribuição de Poisson a quantidade de ocorrências de um evento num intervalo de tempo é independente da quantidade de ocorrências do evento em qualquer outro intervalo distinto.

A fórmula da distribuição de Poisson é definida por: $P(x) = \frac{\left(\lambda^2 \cdot e^{-\lambda}\right)}{x!}$,onde x é a quantidade de ocorrências do evento num determinado intervalo de tempo e λ é a taxa de ocorrência do evento x.

A normalização euclidiana (ou comprimento euclidiano) num vetor u = ($u1$, $u2$, ..., un) na dimensão Rn, é definida por: $\|u\| = (u * u)^{\frac{1}{2}} = \sqrt{u_1^2 + u_2^2 + ... + u_n^2}$

Através destes dois conceitos matemáticos, o Terrier realiza a implementação do PL2 como:

$$NORM \cdot q \cdot \left\{TF \cdot \log_2\left(\frac{1}{fn}\right) + [fn \cdot \log_2(e)] + [0,5 \cdot \log_2(2 \cdot \pi \cdot TF)] + [TF \cdot \log_2(TF) - \log_2(e)]\right\}$$

sendo que:

$$TF = f \cdot log_2\left(1 + \frac{[c \cdot ADL]}{docLength}\right)$$

onde, f é a frequência dos termos dentro do documento, c é uma constante de normalização, ADL é a média do comprimento dos documentos e $docLength$ é o comprimento do documento. $NORM$ é definida pela razão de 1 pelo TF somado a 1 e, por fim, fn é a razão entre a frequência dos termos na coleção de documentos e o número de documentos.

- DFRee (DFR): Modelo hipergeométrico que leva em média duas medidas de informação.

O DFRee é um algoritmo que segue o paradigma do DFR, e tem como objetivo calcular a média do número de bits extra para codificar um token dos termos fornecidos para a consulta de acordo com o DFR. Há duas formas para provar o conceito da probabilidade. O primeiro consiste em considerar apenas um documento da coleção, e o segundo consiste em colher uma amostra do documento de acordo com as estatísticas da coleção inteira. Assim o DFRee realiza

o cálculo da média entre as medidas dos dois documentos, determinado como seu produto interno.

No Terrier a fórmula do DFRee é implementada como:

$$q \cdot norm \cdot \{f \cdot [\log_2(prior \cdot InvPrior)]\} + (f+1) \cdot \log_2(post \cdot InvPrior) + 0,5 \cdot \log_2(post \cdot InvPrior)$$

onde $prior$ é dada pela razão entre a quantidade de termos dentro do documento, f é o comprimento do documento, docLength. Post é definido como: $post = \frac{(f+1)}{docLength+1}$

$InvPrior$ é a razão entre a quantidade de tokens dentro de um documento e a frequência do termo dentro da coleção e $norm$ é igual ao produto da frequência de termos dentro de um documento pelo log na base 2 da razão entre $post$ pelo $prior$.

d. Explique os modelos de ponderação dependentes do campo.

O Terrier tem suporte para modelos de ponderação baseados em campo. Particularmente, os modelos baseados em campo levam em consideração não apenas a presença de um termo num campo, mas a frequência real da ocorrência nesse campo. Por exemplo, para um documento em que o termo da consulta ocorre uma vez no corpo do texto, há apenas uma pequena probabilidade de que o documento esteja realmente relacionado a esse termo. No entanto, se o termo ocorrer no título do documento, essa chance aumenta muito. O Terrier oferece vários modelos de ponderação baseados em campo:

- PL2F: modelo de normalização por campo, baseado no PL2 (anteriormente explicado).
- BM25F: modelo de normalização por campo, baseado no BM25 (anteriormente explicado).
- ML2: modelo baseado em campo multinomial.

A distribuição multinominal é uma generalização da distribuição binomial. Para n ensaios independentes, cada um conduz a um sucesso das k categorias, cada categoria tem probabilidade de sucesso fixa. A distribuição multinominal dá a probabilidade de qualquer combinação particular de número de sucessos para as diferentes categorias.

A distribuição de probabilidade Multinomial, com n número de ensaios e o vetor de probabilidade $p = (p_1, \ldots, p_k)$,é dada por:

$$\mathbb{P}(X_1 = x_1, \ldots, X_k = x_k) = \frac{n!}{x_1! \times \ldots \times x_k!} \times p_1^{x_1} \times p_2^{x_2} \times \ldots \times p_k^{x_k}.$$

- MDL2: modelo baseado em campo multinominal, onde o multinominal é substituído por uma aproximação.

- Modelos arbitrários de ponderação de normalização por campo podem ser gerados usando PerFieldNormWeightingModel de maneira semelhante a DFR Weighting Model (maneira alternativa de especificar um modelo de ponderação DFR arbitrário).

Diferentes modelos baseados em campo têm diferentes parâmetros, controlados por várias propriedades, as quais incluem pesos para cada campo, ou seja, w.0, w.1, etc. Modelos de normalização por campo, como o BM25F e o PL2F também requerem os parâmetros de normalização para cada campo, ou seja, c.0, c.1, etc.

Para executar, na bash, um modelo baseado em campo (neste caso, PL2F):

```
bin/terrier batchretrieval -w PL2F -Dc.0=1.0 -Dc.1=2.3 -Dc.3=40 -Dw.0=4 -Dw.1=2 -Dw.3=25
```

```
################################################
# field-based models
################################################
#Field based models take the frequency in each field
#into account.
#trec.model=BM25F
#trec.model=PL2F
#For these models, you need a length normalisation property
#for each field, starting at 0. Note that field names (as specified by
FieldTags.process)
#are NOT used here.
#c.0=1
#c.1=1
#etc

#Similarly, each field needs a weight. Some models may implicitly have a
restriction
#on these weights (e.g. sum to 1)
#w.0=1
#w.1=1
#etc
```

e. Pode-se inflacionar a pontuação dos documentos baseados na proximidade dos termos da interrogação? Como?

Sim. Supondo que se tem uma query com 2 palavras e há um index com posição de informação. Pode-se dar mais pontuação a palavras que façam parte da query que estejam mais próximas ou não. Há regras de pontuação para ver se queries estão na mesma frase, ou no mesmo parágrafo. Nestes casos, a ordem das palavras não importa. Assim, a distância é importante para a pontuação.

O Terrier inclui dois modelos de dependência. Estes modelos têm grande peso em documentos onde os termos de consulta estão próximos. Para usar um modelo de dependência de termo, deve-se indexar usando blocos. Estes dois modelos são:

- DFRDependenceScoreModifier - implementa um modelo de dependência baseado na divergência da aleatoriedade.
- MRFDependenceScoreModifier - implementa o modelo de dependência de Markov Random Field (é um conjunto de variáveis aleatórias que possuem uma propriedade de Markov descrita por um grafo não-direcionado).

Para ativar os modelos de dependência:

```
#choose between DFR or Markov Random Fields dependence models
#matching.dsms=DFRDependenceScoreModifier
#matching.dsms=MRFDependenceScoreModifier
```

f. Como se pode alterar no modo batch o formato de entrada das interrogações e saídas dos resultados?

TRECQuerying é a principal forma pela qual a recuperação de informação é implantada para experiências de Batch Retrieval.

Existem uma infinidade de formas pelas quais pode ser implementado:

• Formato dos tópicos de entrada: O Terrier oferece suporte a tópicos em dois formatos (com tag TREC ou uma consulta por linha). Se nenhum deles for adequado, pode-se implementar outro QuerySource que saiba como analisar os ficheiros de tópicos. Para tal, usa-se a propriedade trec.topics.parser para se poder usar o novo QuerySource.

Ex: trec.topics.parser = my.package.DBTopicsSource

```
#the default QuerySource, TRECQuery, parses TREC notation topic files
#trec.topics.parser=TRECQuery
```

• Formato dos resultados de saída: pode-se implementar outro OutputFormat para alterar o formato dos resultados nos ficheiros ".res". Usa-se a propriedade trec.querying.outputformat para usar o novo OutputFormat.

Ex: trec.querying.outputformat=my.package.MyTRECResultsFormat

```
#you can override the format of the results file:
#trec.querying.outputformat=my.package.some.other.OutputFormat
```

O comando "terrier/batchretrieve" tem um argumento de linha de comandos: "-s", que permite alterar a entrada e saída dos resultados.

g. Como incluiria na pontuação dos documentos um valor independente da interrogação como PageRank? Explique

O Terrier pode integrar facilmente um recurso de pontuação dos documentos que é independente da query. Para tal, recorre-se ao SimpleStaticScoreModifier.

Pode-se adicionar este recurso:

```
bin/terrier batchretrieval -Dmatching.dsms=SimpleStaticScoreModifier
-Dssa.input.file=/path/to/feature -Dssa.input.type=listofscores -Dssa.w=0.5
```

A propriedade ssa.w controla o peso do recurso. Por fim, o Terrier pode suportar vários DSMs, usando-os de forma delimitada por vírgulas:

```
bin/terrier batchretrieval
-Dmatching.dsms=DFRDependenceScoreModifier,SimpleStaticScoreModifier
-Dssa.input.file=/path/to/feature -Dssa.input.type=listofscores -Dssa.w=0.5
```

O PageRank é um algoritmo de ordenação muito usado entre os motores de busca e fez parte dos algoritmos de ordenação da Google. Sendo um algoritmo estático e dependente da estrutura web, consiste em basicamente levar em consideração que uma página possui vários links a apontar para ela, levando também em consideração que o utilizador nunca irá voltar para uma página que já foi explorada, e, desta forma, a ordenação da página é calculada de acordo com a ordenação das páginas anteriores e a quantidade de backlinks entrando na página. Isto é, se uma página atual é referenciada por vários links de páginas com ordenação alta e se essa mesma página possui grande quantidade de backlinks, então essa página possuirá uma ordenação muito alta.

Tal como as páginas com ordenação alta são importantes e relevantes, assim as páginas com backlinks para a página atual também serão consideradas importantes e relevantes.

Assim é possível definir o PageRank, PR(a) como: $PR(a) = q + (1-q)\sum_{i=1}^{n} \frac{PR(p_i)}{C(p_i)}$,onde q deve ser definido pelo sistema, geralmente é atribuído o valor de 0,15. Esta fórmula permite que a ordenação de outras páginas seja normalizada pelo número de links na página.

Uma forma simplificada de cálculo de PageRank $PR(a) = \sum_{v \epsilon B_a} \frac{PR(v)}{L(v)}$,onde o valor do PageRank de uma página "a", $PR(a)$, depende dos valores do PageRank de cada página "v", $PR(v)$, fora do conjunto de páginas que apontam para a página "a", Ba, dividido pelo número de links que vem da página "v".

h. Que outro tipo de configuração pode ser feita que não foi apresentada?

Uma configuração possível é alterar o número máximo de documentos retornados. Para cada query, o Terrier retorna um número máximo de 1000 documentos, por default. Poder-se-á alterar este valor através da propriedade "matching.retrieved_set_size". Por exemplo, se o objetivo for recuperar 10.000 documentos para cada query, será necessário definir "matching.retrieved_set_size=10.000". Além disso, se o endcontrole (que define o último resultado a ser recuperado (por omissão 0 - mostrar todos os resultados) estiver definido na propriedade querying.default.controls, altera-se para 9999.

Outra configuração relaciona-se com os modelos de ponderação. Alguns destes modelos como por exemplo, BM25, assumem baixa frequência de documentos de termos de query. Para estes modelos, vale a pena ignorar os termos de query com alta frequência de documento durante a recuperação, definindo "ignore.low.idf.terms=True". Além disto, é também vantajoso definir "ignore.low.idf.terms=false" para tarefas de pesquisa de alta precisão.

4) Explique sucintamente a linguagem de interrogação disponível no Terrier.

O Terrier oferece duas linguagens de interrogação: uma linguagem de interrogação de alto nível voltada para o utilizador, e uma linguagem de interrogação de baixo nível para programadores que é expressa em termos de operações de correspondência (matching ops).

A linguagem de interrogação de alto nível do Terrier é flexível ao utilizador para pesquisar frases, campos ou termos em documentos.

Alguns exemplos da linguagem de interrogação do Terrier são os seguintes:

- term1 term2 - recupera documentos que contêm term1 ou term2 (não precisa conter ambos).
- {term1 term2} - recupera documentos que contêm term1 ou term2, onde são tratados como sinónimos um do outro.
- term1^2.3 - o peso do term1 é multiplicado por 2,3.
- +term1 +term2 - recupera os documentos que contêm term1 e term2.
- +term1 -term2 - recupera documentos que contêm term1 e não contêm term2.
- title:term1 - recupera os documentos que contêm term1 no campo do título (a indexação de campo deve ser configurada para registrar o campo de título).
- +title:term1 - recupera os documentos que contêm term1 no campo de título. Tal como acima, a indexação de campo deve estar ligada.

- title:(term1 term2)^2.3 – recupera os documentos que contêm term1 ou term2 no campo de título e aumenta seus pesos em 2,3. A indexação de campo deve estar ligada.
- term1 -title:term2 – recupera os documentos que contêm term1, mas não devem conter term2 no campo de título.
- "term1 term2" - recupera os documentos onde os termos term1 e term2 aparecem numa frase. O Terrier considera isto equivalente a +"term1 term2".
- "term1 term2"~n - recupera os documentos onde os termos term1 e term2 aparecem a uma distância de n blocos. A ordem dos termos não é considerada.

Pode-se fazer combinações entre os diferentes exemplos acima referidos. Para utilizar a linguagem de consulta do Terrier neste caso, deve-se usar SingleLineTRECQuery e definir no ficheiro "terrier.properties" a propriedade: "SingleLineTRECQuery.tokenise" como "false".

A linguagem de baixo nível para programadores do Terrier (Matching Op Query Language), segue um subconjunto da linguagem de query Indri. Há dois tipos de operadores: semânticos, no sentido de que fazem com que uma posting list com uma semântica particular seja gerada no momento da correspondência; em contraste, os operadores sintáticos são "syntactic sugar" definidos na linguagem de query, que permitem que os atributos dos operadores semânticos sejam alterados.

Os operadores semânticos são:

- term1 - pontua documentos que contêm este único termo de query.
- term1.title - pontua documentos que contêm este único termo de query no título.
- #band(op1 op2) - pontua um termo que contém ambos os operadores. A frequência de cada documento correspondente é 1.
- #syn(op1 op2) - pontua documentos que contêm op1 ou op2. A frequência de cada documento correspondente é a soma das frequências das palavras constituintes.
- #prefix(term1) - pontua documentos que contêm termos prefixados por term1.
- #fuzzy(term1)- pontua documentos que contêm termos que combinam vagamente com o termo1.
- #uw8(op1 op2) - o operador #uwN pontua documentos op1 ou op2 dentro de janelas não ordenadas de N tokens - neste caso, janelas de 8 tokens de tamanho.
- #1(op1 op2) - o operador nº 1 pontua os documentos op1 ou op2 que aparecem adjacentes.
- #band(op1 op2) - o operador #band pontua documentos que contêm op1 e op2.
- #base64(term1) - permite que uma representação base64 de um termo de query seja expressa, o qual não é diretamente compatível com o "matchop ql".

Atualmente, existem dois operadores sintáticos:

- #tag(tagName op1 op2) - define o atributo de tag desses termos de query.
- #combine:k=v(op1 op2) - o #combineoperador permite que vários operadores sejam agrupados. Além disto, vários pares de valores-chave podem ser especificados para controlar esses termos de query. Em particular, o peso de um termo pode ser controlado definindo o índice desse operador - por exemplo #combine:0=2:1=1(op1 op2), definirá duas vezes mais peso em op1 do que em op2. O modelo de ponderação (wmodel) e tag (tag) do(s) termo(s) de query também podem ser definidos, por exemplo: #combine:tag=second:wmodel=PL2(op1 op2).

Nota: Operadores semânticos não podem conter operadores sintáticos.

Para usar a "Matching Op Query Language" na linha de comandos:

```
$ bin/terrier interactive -m
Setting TERRIER_HOME to /home/Terrier
23:33:14.496 [main] INFO  o.t.structures.CompressingMetaIndex - Structure meta reading lookup file into memory
23:33:14.503 [main] INFO  o.t.structures.CompressingMetaIndex - Structure meta loading data file into memory
matchop query> #combine:0=0.85:1=0.15:2=0.05(#combine(dramatise personae) #1(dramatise personae) #uw8(dramatise personae))
etc
```

5) Crédito Extra: Explique o que é a expansão de interrogações e como está implementada no Terrier.

Os termos da expansão das interrogações correspondem aos termos mais informativos dos documentos principais. Neste processo de expansão, os termos nos documentos mais devolvidos são ponderados usando um modelo de ponderação de DFR específico. Atualmente, o Terrier implementa os modelos de ponderação de termo Bo1 (Bose-Einstein 1), Bo2 (Bose-Einstein 2) e KL (Kullback-Leibler). Os modelos de ponderação de termo DFR seguem uma abordagem livre de parâmetros por default.

Uma abordagem alternativa é o mecanismo de expansão de interrogações de Rocchio. Um utilizador pode mudar para a última abordagem definindo "parameter.free.expansion" como false no ficheiro "terrier.properties". O valor padrão do parâmetro beta da abordagem de Rocchio é 0,4. Para alterar este parâmetro, o utilizador precisa especificar a propriedade rocchio_beta no ficheiro "terrier.properties".

Além disto, existem dois parâmetros que podem ser definidos para aplicar a expansão de interrogações. O primeiro é o número de termos com os quais se pretende expandir uma query, especificado pela propriedade "expansion.terms- ", sendo o valor default de 10. Além

disso, o número de documentos com melhor classificação dos quais esses termos são extraídos é especificado pela propriedade "expansion.documents", cujo valor default é 3.

Para recuperar de uma coleção de teste indexada, usando a expansão de interrogações e com o parâmetro de normalização de frequência do termo igual a 1,0, pode-se implementar:

```
bin/terrier batchretrieval -q -c c:1.0
```

Para usar documentos de feedback na expansão da consulta, altere o "FeedbackSelector" da seguinte forma:

```
bin/terrier batchretrieval -q
-Dqe.feedback.selector=RelevantOnlyFeedbackDocuments,RelevanceFeedbackSelector
-Dqe.feedback.filename=/path/to/feedback/qrels
```

6) Avaliação

a. Que ferramentas tem disponível no Terrier para avaliação

O Terrier usa o jtreceval para fornecer a avaliação de execuções de recuperação standards. Este contém o trec_eval compilado para rodar em várias plataformas (por exemplo, Windows / Linux / Mac x86). Para plataformas não que não o suportam, o Terrier também possui um pacote de avaliação Java para avaliar resultados de tarefas “TREC ad hoc” e de localização de páginas.

Outras formas de avaliação são: CLEF, NTCIR ou FIRE.

Normalmente, para avaliar a eficácia da recuperação ao usar a expansão de queries, cada query é executada com e sem expansão e, em seguida, os dois conjuntos de resultados recuperados são comparados.

As duas medidas mais comuns usadas para avaliar a eficácia da recuperação são: a precisão média para cada query, e a MAP (média de todas as precisões) para um grupo de queries.

Precisão Média é a soma da precisão em cada documento relevante do conjunto de resultados, dividido pelo número de documentos relevantes. MAP é calculado pela soma da precisão média obtida para cada query, dividido pelo número de tópicos submetidos ao sistema.

No contexto dos próprios sistemas de recuperação de informação, o resultado das consultas pode ser um conjunto de milhares de documentos relevantes, e os utilizadores não estarão interessados em obter todos eles. Assim, a precisão em k documentos é útil.

b. Os Runs criados pelo Terrier são compatíveis com a ferramenta trec_eval. Descreva sucintamente as funcionalidades desta ferramenta. Que resultados permite obter?

O trec_eval é a ferramenta standard usada pela comunidade TREC para avaliar uma execução de recuperação "ad hoc", dado o ficheiro de resultados e um conjunto padrão de resultados avaliados.

Todas as opções desta funcionalidade estão descritas em: "bin/terrier help trec_eval"

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier help trec_eval
Terrier version 5.4
trec_eval [-h] [-q] [-m measure[.params] [-c] [-n] [-l <num>]
   [-D debug_level] [-N <num>] [-M <num>] [-R rel_format] [-T results_format]
   rel_info_file  results_file
```

Antes de executar uma avaliação, é preciso especificar o ficheiro de avaliações de relevância na propriedade trec.qrels.

```
# Evaluation
################################################
trec.qrels=/Users/macedo/Terrier_eclipse/myTerrier/Outros/qrels/qrelTopics201_350.txt
#trec.qrels=/Users/macedo/Terrier_eclipse/myTerrier/Outros/qrels/qrels_ah_PT-2005.txt
#trec.qrels=/Users/macedo/Terrier_eclipse/myTerrier/Outros/qrels/qrels_ah_PT-2005-2006.txt
```

Para avaliar todos os ficheiros de extensão ".res" na pasta "var/results", pode-se digitar:

```
bin/terrier batchevaluate
```

O comando "batchevaluate" avalia cada ficheiro usando o trec_eval. Pode-se avaliar um ficheiro de uma determinada extensão fornecendo o nome do ficheiro na linha de comandos:

```
   bin/terrier batchevaluate -e InL2c1.0_0.res
or
   bin/trec_terrier.sh -e ./var/results/InL2c1.0_0.res
```

O comando acima avalia apenas o ficheiro "./var/results/InL2c1.0_0.res". Para um ficheiro de resultados denominado "x.res", o resultado da avaliação é guardado no ficheiro x.eval, que contém o conteúdo conforme mostrado no exemplo a seguir:

```
Number of queries  = 100
Retrieved          = 100000
Relevant           = 3422
Relevant retrieved = 1687
---------------------------
Average Precision: 0.1459
R Precision      : 0.1485
---------------------------
Precision at   1 : 0.2400
Precision at   2 : 0.2450
Precision at   3 : 0.2367
Precision at   4 : 0.2250
Precision at   5 : 0.2160
Precision at   10 : 0.1870
Precision at   15 : 0.1647
Precision at   20 : 0.1515
Precision at   30 : 0.1280
Precision at   50 : 0.0956
Precision at   100 : 0.0614
Precision at   200 : 0.0408
Precision at   500 : 0.0243
Precision at   1000 : 0.0169
---------------------------
Precision at   0%: 0.3501
Precision at   10%: 0.3027
Precision at   20%: 0.2502
Precision at   30%: 0.2015
Precision at   40%: 0.1746
Precision at   50%: 0.1449
Precision at   60%: 0.1074
Precision at   70%: 0.0859
Precision at   80%: 0.0655
Precision at   90%: 0.0358
Precision at   100%: 0.0194
---------------------------
Average Precision: 0.1459
```

As medidas de avaliação exibidas acima têm a média de um conjunto de queries. Podemos obter resultados por query usando a opção -p na linha de comandos:

```
bin/terrier batchevaluate -e PL2c1.0_0.res -p
```

A saída resultante guardada no ficheiro de extensão ".eval" poderá conter outros resultados.

Poder-se-á usar facilmente o trec_eval diretamente na linha de comandos, invocando o script "trec_eval.sh":

```
bin/terrier trec_eval /path/to/qrels var/results/PL2c1.0_0.res
```

II. Experiências com a Coleção Chave

Primeiramente, começa-se por abrir o Terrier, como na foto abaixo:

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>trec_setup.bat /Users/paulo/Documents/terrier-project-5.4/share-pt/publ
ico-npl/corpus
Set TERRIER_HOME to be C:\Users\paulo\Documents\terrier-project-5.4
TERRIER_HOME is set to C:\Users\paulo\Documents\terrier-project-5.4
ECHO is off.
Creating collection.spec file.
Creating jforests.properties file.
Creating features.list file.
Creating logging configuration (logback.xml) file in C:\\Users\\paulo\\Documents\\terrier-project-5.4\etc\
Creating terrier.properties file.
Now building collection.spec


Updated collection.spec file. Please check that it contains all and only all
files to indexed, or create it manually

collection.spec:
--------------------------------------------------------------------------------
#add the files to index
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940101.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940102.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940103.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940104.sgml
```

1) Indexe a coleção Público incluída na Chave

De modo a indexar apenas a coleção público, é necessário ter no ficheiro “etc/collection.spec” todos os PATHs referentes aos ficheiros, tal como mostrado abaixo.

collection.spec

```
#add the files to index
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940101.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940102.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940103.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940104.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940105.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940106.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940107.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940108.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940109.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940110.sgml
C:\Users\paulo\Documents\terrier-project-5.4\share-pt\publico-npl\corpus\ED940111.sgml
```

a. Quanto tempo demora a indexar o público com a indexação dum passo com e sem informação de posição?

Na bash, para indexar o que está na “collection.spec”, que neste caso corresponde à coleção público, usando a indexação de apenas um passo, usou-se o seguinte comando:

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchindexing -j
```

COM INFORMAÇÃO DE POSIÇÃO:

Para indexar com informação de posição, no ficheiro "etc/terrier.properties" tem que se "ativar", ou seja, colocar como *True* a seguinte opção:

```
#To record term positions (blocks) in the index, set
block.indexing=true
```

Supõe-se que indexar com informação de posição é mais lento do que indexar sem informação de posição.

Após correr na bash o comando acima para indexação em 1 passo, tem-se que:

```
20:52:58.406 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 took 83 seconds to build the runs for 106821 documents

20:52:58.748 [main] INFO  o.t.structures.indexing.Indexer - Merging 1 runs...
20:53:06.966 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 took 8 seconds to merge

20:53:06.967 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 total time 91
20:53:06.970 [main] INFO  o.t.s.indexing.LexiconBuilder - Optimising structure lexicon
20:53:06.972 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - Optimising lexicon with 283229 entries
20:53:07.224 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - All ids for structure lexicon are aligned, skipping .fsomapid file
Total time elaped: 93 seconds
```

O Terrier demorou **93** segundos para realizar o indexamento com informação de posição, a 1 passo, por completo.

SEM INFORMAÇÃO DE POSIÇÃO:

Para indexar sem informação de posição, no ficheiro "etc/terrier.properties" tem que se "desativar", ou seja, colocar como *False* a seguinte opção, ou apenas comentar a linha:

```
#To record term positions (blocks) in the index, set
#block.indexing=true
```

Após correr na bash o comando acima para indexação em 1 passo, tem-se que:

```
20:55:14.168 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 took 66 seconds to build the runs for 106821 documents

20:55:14.530 [main] INFO  o.t.structures.indexing.Indexer - Merging 1 runs...
20:55:17.040 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 took 2 seconds to merge

20:55:17.041 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 total time 68
20:55:17.043 [main] INFO  o.t.s.indexing.LexiconBuilder - Optimising structure lexicon
20:55:17.045 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - Optimising lexicon with 283229 entries
20:55:17.321 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - All ids for structure lexicon are aligned, skipping .fsomapid file
Total time elaped: 70 seconds
```

O Terrier demorou **70** segundos para realizar o indexamento sem informação de posição, a 1 passo, por completo.

Tal como suposto anteriormente, comprovou-se que indexar com informação de posição é mais lento do que indexar sem informação de posição (93 segundos > 70 segundos), o que faz sentido uma vez que para colocar a informação de posição é necessário mais tempo.

b. Quanto tempo demora a indexar o público com a indexação de dois passos com e sem informação de posição?

Na bash, para indexar o que está na "collection.spec", que neste caso corresponde à coleção público, usando a indexação de dois passo, usou-se o seguinte comando:

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchindexing
```

Para indexar com e sem informação de posição usam-se as mesmas propriedades apresentadas acima, na alínea a). E espera-se chegar à mesma conclusão. Além disto, espera-se poder concluir que a indexação a 2 passos é mais lenta que a indexação a 1 passo.

COM INFORMAÇÃO DE POSIÇÃO:

```
20:49:40.113 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 took 121seconds to index (106821 documents)

20:49:40.365 [main] INFO  o.t.s.indexing.LexiconBuilder - 2 lexicons to merge
20:49:41.055 [main] INFO  o.t.s.indexing.LexiconBuilder - Optimising structure lexicon
20:49:41.057 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - Optimising lexicon with 283229 entries
20:49:41.844 [main] INFO  o.t.structures.indexing.Indexer - Started building the block inverted index...
20:49:41.860 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - creating block inverted index
20:49:41.866 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - BlockMemSizeLexiconScanner: lexicon scanning until approx 3 GiB of memory, including positions, is consumed
20:49:41.872 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - Iteration 1 of 1 (estimated) iterations
20:49:43.139 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - time to process part of lexicon: 1.266
20:49:43.146 [main] INFO  o.t.structures.FSADocumentIndex - Document index requires 417,3 KiB remaining heap is 3,2 GiB
20:49:43.158 [Thread-1] INFO  org.terrier.utility.TerrierTimer - Loading document document lengths 2% done. Estimated finished in 0m00s
20:50:08.810 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - time to traverse direct file: 25.669
20:50:14.681 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - time to write inverted file: 5.87
20:50:14.682 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - temporary memory used: 569,2 MiB
20:50:14.683 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - time to perform one iteration: 32.811
20:50:14.684 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - number of pointers processed: 29165780
20:50:14.692 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - Finished generating inverted file, rewriting lexicon
20:50:15.317 [main] INFO  o.t.s.indexing.LexiconBuilder - Optimising structure lexicon
20:50:15.319 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - Optimising lexicon with 283229 entries
20:50:15.726 [main] INFO  o.t.structures.indexing.Indexer - Finished building the block inverted index...
20:50:15.729 [main] INFO  o.t.structures.indexing.Indexer - Time elapsed for inverted file: 33
Total time elaped: 157 seconds
```

O Terrier demorou **157** segundos para realizar o indexamento com informação de posição, a 2 passos, por completo.

SEM INFORMAÇÃO DE POSIÇÃO:

```
20:58:16.012 [main] INFO  o.t.structures.indexing.Indexer - Collection #0 took 94 seconds to index (106821 documents)
20:58:16.557 [main] INFO  o.t.s.indexing.LexiconBuilder - 2 lexicons to merge
20:58:17.175 [main] INFO  o.t.s.indexing.LexiconBuilder - Optimising structure lexicon
20:58:17.178 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - Optimising lexicon with 283229 entries
20:58:17.757 [main] INFO  o.t.structures.indexing.Indexer - Started building the inverted index...
20:58:17.792 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - BasicMemSizeLexiconScanner: lexicon scanning until approx 3 GiB of memory is consumed
20:58:17.802 [main] INFO  o.t.s.i.c.InvertedIndexBuilder - Iteration 1 of 1 (estimated) iterations
20:58:31.993 [main] INFO  o.t.s.indexing.LexiconBuilder - Optimising structure lexicon
20:58:31.993 [main] INFO  o.t.s.i.FSOMapFileLexiconUtilities - Optimising lexicon with 283229 entries
20:58:32.329 [main] INFO  o.t.structures.indexing.Indexer - Finished building the inverted index...
20:58:32.329 [main] INFO  o.t.structures.indexing.Indexer - Time elapsed for inverted file: 14
Total time elaped: 110 seconds
```

O Terrier demorou **110** segundos para realizar o indexamento sem informação de posição, a 2 passos, por completo.

Para concluir, verifica-se que, tal como na indexação a 1 passo, na indexação a 2 passos indexar com informação de posição é mais lento do que indexar sem informação de posição, pelo mesmo motivo referido anteriormente.

Além disto, verifica-se também, tal como previsto, que a indexação em 1 passo é mais rápida que a indexação em 2 passos, quer se indexe com ou sem informação de posição.

2) Vamos usar o Terrier com a coleção Público e o conjunto de interrogações e juízos de relevância disponibilizados no CLEF. Os resultados vão ser obtidos com 100 interrogações.

As propriedades usadas neste exercício, à exceção da alínea d), estão apresentadas abaixo.

```
#default and allowed controls
querying.default.controls=wmodel:DPH,parsecontrols:on,parseql:on,applypipeline:on,terrierql:on,localmatching:on,filters:on,decorate:on
querying.allowed.controls=scope,qe,qemodel,start,end,site,scope


#To record term positions (blocks) in the index, set
#block.indexing=true

#document tags specification
#for processing the contents of
#the documents, ignoring DOCHDR
#TrecDocTags.doctag=DOC
#TrecDocTags.idtag=DOCNO
#TrecDocTags.skip=DOCHDR

#the documents, ignoring tags in TrecDocTags.skip
TrecDocTags.doctag=DOC
TrecDocTags.idtag=DOCNO
TrecDocTags.skip=DOCID
TrecDocTags.process=DOC,DOCNO,DATE,CATEGORY,TEXT


#set to true if the tags can be of various case
TrecDocTags.casesensitive=false
#starting from Terrier 5.3, we assume that documents are in UTF-8
trec.encoding=UTF-8

#query tags specification
TrecQueryTags.doctag=top
TrecQueryTags.process=top,num,PT-title,PT-desc
TrecQueryTags.idtag=num
TrecQueryTags.skip=PT-narr

# Information Retrieval model to use
# can be more than one
#trec.model=PL2
#trec.model=TF_IDF

#stopwords.filename=/Users/paulo/Documents/terrier-project-5.4/share-pt/stopword-list.txt
#the processing stages a term goes through
#termpipelines=PortugueseSnowballStemmer
#termpipelines=Stopwords,PortugueseSnowballStemmer
termpipelines=


# Meta Index s? com docno
indexer.meta.forward.keys=docno
indexer.meta.forward.keylens=20
indexer.meta.reverse.keys=docno

# Evaluation
################################################
trec.qrels=/Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/qrels
```

a. Vamos fazer um gráfico de precisão e recall, r-precision, MAP aos 10,20 e 30.

Após a indexação, realizada para este exercício em dois passos e sem informação de posição, correu-se o seguinte comando – "terrier batchretrieve -t <path da query-trec.trec>":

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchretrieve -t /Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/query-text.trec
```

Depois de feito o batchretrieve, procedeu-se à realização do "batchevaluate -j", que, além de mostrar um valor de precisão, cria um ficheiro "_.eval", com o resultado da avaliação, neste caso usando, por defeito, o modelo DPH.

Assim, observando a foto abaixo, pode-se concluir que, através do "batchretrieve" foi criado o ficheiro "var/results/DPH_0.res", e após feita a avaliação, a precisão foi de **0.3605**.

```
19:54:57.748 [main] INFO  o.t.matching.PostingListManager - Query C350 with 15 terms has 15 posting lists
19:54:57.807 [main] INFO  org.terrier.querying.LocalManager - running process PostFilterProcess
19:54:57.811 [main] INFO  org.terrier.querying.LocalManager - Finished executing query C350 in 78ms - 1000 results retrieved
19:54:57.813 [main] INFO  o.t.a.batchquerying.TRECQuerying - Time to process query C350: 0.08
19:54:57.831 [main] INFO  o.t.a.batchquerying.TRECQuerying - Settings of Terrier written to C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_0.res.settings
19:54:57.838 [main] INFO  o.t.a.batchquerying.TRECQuerying - Finished topics, executed 100 queries in 10.747 seconds, results written to C:\Users\paulo\Documents\terrier-pro
ject-5.4\var\results/DPH_0.res

C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchevaluate -j
19:55:05.582 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_0.res
Average Precision: 0.3605
```

Do "batchevaluate -j" resulta também o ficheiro "DPH_0.eval", ficheiro este apresentado abaixo, e de onde se retiram os valores pedidos tal como os dados para a realização do gráfico pedido.

DPH_0.eval

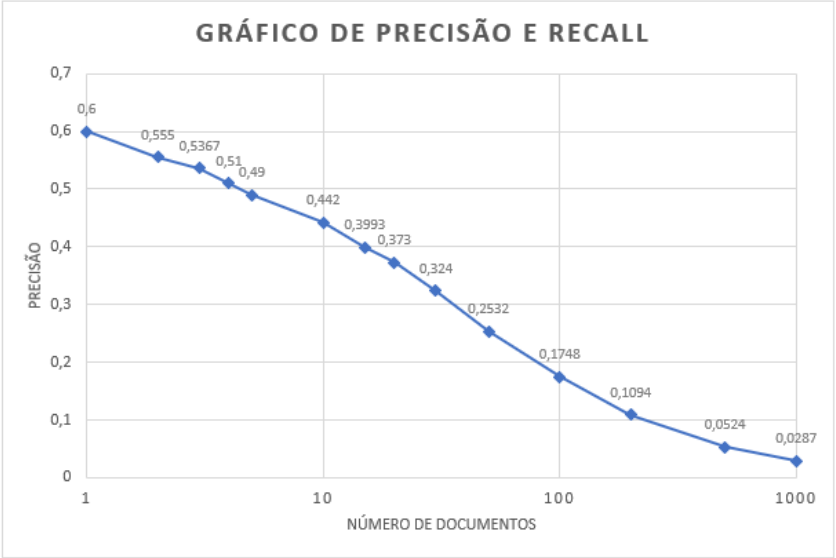
```
Number of queries  = 100
Retrieved          = 100000
Relevant           = 3422
Relevant retrieved = 2871

Average Precision: 0.3605
R Precision      : 0.3699

Precision at   1 : 0.6000
Precision at   2 : 0.5550
Precision at   3 : 0.5367
Precision at   4 : 0.5100
Precision at   5 : 0.4900
Precision at   10 : 0.4420
Precision at   15 : 0.3993
Precision at   20 : 0.3730
Precision at   30 : 0.3240
Precision at   50 : 0.2532
Precision at   100 : 0.1748
Precision at   200 : 0.1094
Precision at   500 : 0.0524
Precision at   1000 : 0.0287

Precision at   0%: 0.7520
Precision at   10%: 0.6336
Precision at   20%: 0.5399
Precision at   30%: 0.4717
Precision at   40%: 0.4255
Precision at   50%: 0.3813
Precision at   60%: 0.2963
Precision at   70%: 0.2524
Precision at   80%: 0.1992
Precision at   90%: 0.1495
Precision at   100%: 0.0974

Average Precision: 0.3605
```

MAP: 0.3605

R Precision: 0.3699

Com os dados de precisão de 1 a 1000 documentos que estão no ficheiro representado acima, fez-se um gráfico de **precisão e recall.** É de notar que a escala do eixo do X é uma escala logarítmica, de modo a que os dados, no gráfico, fiquem mais distribuídos.

DOCS	PRECISION
1	0,6
2	0,555
3	0,5367
4	0,51
5	0,49
10	0,442
15	0,3993
20	0,373
30	0,324
50	0,2532
100	0,1748
200	0,1094
500	0,0524
1000	0,0287



b. Vamos comparar dois métodos de ponderação disponibilizados no Terrier! (por exemplo! TF-IDF e BM25) e em particular um baseado no DFR.

Nesta alínea vai-se analisar o método default – DPH, usado na alínea a), que é baseado no DFR, o método BM25 e o método TF_IDF.

Em primeiro lugar realizou-se o "batchretrieve" do modelo BM25, e de seguida fez-se a respetiva avaliação. Foram criados os ficheiros "BM25_1.res" e "BM25_1.eval", respetivamente.

- *"bin/terrier batchretrieve -w BM25 -c c:0.4 -t <path da query-trec.trec>"*
- *"bin/terrier batchevaluate -f -j"*

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchretrieve -w BM25 -c c:0.4 -t /Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/query-text.trec
```

Após isto, fez-se o mesmo para o modelo TF_IDF. Foram criados os ficheiros "TF_IDF_2.res" e "TF_IDF_2.eval", respetivamente.

- *"bin/terrier batchretrieve -w TF_IDF -t <path da query-trec.trec>"*
- *"bin/terrier batchevaluate -f -j"*

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchretrieve -w TF_IDF -t /Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/query-text.trec
```

Como resultado final das avaliações, tem-se:

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchevaluate -f -j
20:02:49.170 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/BM25_1.res
Average Precision: 0.1459
20:02:49.531 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_0.res
Average Precision: 0.3605
20:02:49.717 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/TF_IDF_2.res
Average Precision: 0.3710
```

Além das 3 previsões apresentadas na linha de comandos, foram criados os 2 ficheiros ".eval".

Abaixo está uma representação dos 3 ficheiros, dos 3 modelos, que se pretende comparar.

DPH_0.eval

```
Number of queries   = 100
Retrieved           = 100000
Relevant            = 3422
Relevant retrieved  = 2871

Average Precision: 0.3605
R Precision      : 0.3699

Precision at   1 : 0.6000
Precision at   2 : 0.5550
Precision at   3 : 0.5367
Precision at   4 : 0.5100
Precision at   5 : 0.4900
Precision at   10 : 0.4420
Precision at   15 : 0.3993
Precision at   20 : 0.3730
Precision at   30 : 0.3240
Precision at   50 : 0.2532
Precision at   100 : 0.1748
Precision at   200 : 0.1094
Precision at   500 : 0.0524
Precision at   1000 : 0.0287

Precision at   0%: 0.7520
Precision at   10%: 0.6336
Precision at   20%: 0.5399
Precision at   30%: 0.4717
Precision at   40%: 0.4255
Precision at   50%: 0.3813
Precision at   60%: 0.2963
Precision at   70%: 0.2524
Precision at   80%: 0.1992
Precision at   90%: 0.1495
Precision at   100%: 0.0974

Average Precision: 0.3605
```

BM25_1.eval

```
Number of queries   = 100
Retrieved           = 100000
Relevant            = 3422
Relevant retrieved  = 1687

Average Precision: 0.1459
R Precision      : 0.1485

Precision at   1 : 0.2400
Precision at   2 : 0.2450
Precision at   3 : 0.2367
Precision at   4 : 0.2250
Precision at   5 : 0.2160
Precision at   10 : 0.1870
Precision at   15 : 0.1647
Precision at   20 : 0.1515
Precision at   30 : 0.1280
Precision at   50 : 0.0956
Precision at   100 : 0.0614
Precision at   200 : 0.0408
Precision at   500 : 0.0243
Precision at   1000 : 0.0169

Precision at   0%: 0.3501
Precision at   10%: 0.3027
Precision at   20%: 0.2502
Precision at   30%: 0.2015
Precision at   40%: 0.1746
Precision at   50%: 0.1449
Precision at   60%: 0.1074
Precision at   70%: 0.0859
Precision at   80%: 0.0655
Precision at   90%: 0.0358
Precision at   100%: 0.0194

Average Precision: 0.1459
```

TF_IDF_2.eval

```
Number of queries   = 100
Retrieved           = 100000
Relevant            = 3422
Relevant retrieved  = 2905

Average Precision: 0.3710
R Precision      : 0.3704

Precision at   1 : 0.6000
Precision at   2 : 0.5600
Precision at   3 : 0.5367
Precision at   4 : 0.5025
Precision at   5 : 0.5060
Precision at   10 : 0.4430
Precision at   15 : 0.4073
Precision at   20 : 0.3815
Precision at   30 : 0.3280
Precision at   50 : 0.2616
Precision at   100 : 0.1823
Precision at   200 : 0.1126
Precision at   500 : 0.0534
Precision at   1000 : 0.0290

Precision at   0%: 0.7465
Precision at   10%: 0.6313
Precision at   20%: 0.5565
Precision at   30%: 0.4858
Precision at   40%: 0.4373
Precision at   50%: 0.3939
Precision at   60%: 0.3172
Precision at   70%: 0.2700
Precision at   80%: 0.2183
Precision at   90%: 0.1599
Precision at   100%: 0.0962

Average Precision: 0.3710
```

Analisando o valor das precisões, verifica-se que o método TF_IDF (precisão = 0.3710) é mais preciso que o método DPH_0 (precisão = 0.3605), apesar de ser por uma margem mínima. No entanto, qualquer um destes dois é muito mais preciso que o método BM25_1 (precisão=0.1459).

c. Qual é a melhoria dos resultados se usarmos expansão de interrogações?

De modo a se poder avaliar a melhoria dos resultados, em cada método, do uso da expansão de interrogações, teve que se fazer de novo os 3 "batchretrieve" feitos anteriormente (um para cada método), mas desta vez com "-q" na instrução, de modo a "ativar" esta expansão.

Os 3 *"bin/terrier batchretrieve -q -w <método> -t <path da query-trec.trec>"* realizados estão apresentados abaixo:

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchretrieve -q -t /Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/query-text.trec
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchretrieve -q -w BM25 -c c:0.4 -t /Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/query-text.trec
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchretrieve -q -w TF_IDF -t /Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/query-text.trec
```

Após cada "batchretrieve" é realizado o "batchevaluate -f -q". No final, como resultado, surgem precisões de 6 ficheiros, os 3 anteriores sem expansão de query e os 3 criados agora, de formato "<método>_d_3_t_10_#.res".

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchevaluate -f -j
20:08:26.556 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/BM25_1.res
Average Precision: 0.1459
20:08:26.923 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/BM25_d_3_t_10_4.res
Average Precision: 0.2159
20:08:27.112 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_0.res
Average Precision: 0.3605
20:08:27.353 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_d_3_t_10_3.res
Average Precision: 0.4045
20:08:27.552 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/TF_IDF_2.res
Average Precision: 0.3710
20:08:27.859 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/TF_IDF_d_3_t_10_5.res
Average Precision: 0.4102
```

Daqui podemos concluir que, qualquer que seja o método, com a expansão de interrogações, a precisão é superior:

- Método BM25: a precisão inicial era 0.1459 e, com a expansão, subiu para 0.2159.
- Método DPH: a precisão inicial era 0.3605 e, com a expansão, subiu para 0.4045.
- Método TF_IDF: a precisão inicial era 0.3710 e, com a expansão, subiu para 0.4102.

Pode-se também observar que o que foi concluído anteriormente sem expansão volta aqui a verificar-se, agora com expansão de queries. Ou seja, se anteriormente o método TF_IDF era o mais preciso, aqui comprova-se que, mesmo com a expansão, continua a ter um valor de precisão superior. Da mesma forma, o método BM25 continua a ser o que tem uma precisão mais baixa.

De notar que esta melhoria que se observa através da expansão de interrogações só existe se a indexação for feita, seja qual for o método, sem informação de posição. No caso de a indexação ser feita com informação de posição, a expansão não altera o resultado da precisão.

d. O facto de se usar stopwords e stemming melhora os resultados?

As propriedades usadas nesta alínea foram alteradas em relação às alíneas anteriores, e estão apresentadas abaixo.

```
#default and allowed controls
querying.default.controls=wmodel:DPH,parsecontrols:on,parseql:on,applypipeline:on,terrierql:on,localmatching:on,filters:on,decorate:on
querying.allowed.controls=scope,qe,qemodel,start,end,site,scope


#To record term positions (blocks) in the index, set
#block.indexing=true

#document tags specification
#for processing the contents of
#the documents, ignoring DOCHDR
#TrecDocTags.doctag=DOC
#TrecDocTags.idtag=DOCNO
#TrecDocTags.skip=DOCHDR

#the documents, ignoring tags in TrecDocTags.skip
TrecDocTags.doctag=DOC
TrecDocTags.idtag=DOCNO
TrecDocTags.skip=DOCID
TrecDocTags.process=DOC,DOCNO,DATE,CATEGORY,TEXT


#set to true if the tags can be of various case
TrecDocTags.casesensitive=false
#starting from Terrier 5.3, we assume that documents are in UTF-8
trec.encoding=UTF-8

#query tags specification
TrecQueryTags.doctag=top
TrecQueryTags.process=top,num,PT-title,PT-desc
TrecQueryTags.idtag=num
TrecQueryTags.skip=PT-narr

# Information Retrieval model to use
# can be more than one
#trec.model=PL2
#trec.model=TF_IDF

stopwords.filename=stopword-list.txt
#the processing stages a term goes through
#termpipelines=PortugueseSnowballStemmer
termpipelines=Stopwords,PortugueseSnowballStemmer
#termpipelines=


# Meta Index s com docno
indexer.meta.forward.keys=docno
indexer.meta.forward.keylens=20
indexer.meta.reverse.keys=docno

# Evaluation
################################################
trec.qrels=/Users/paulo/Documents/terrier-project-5.4/share-pt/publico-npl/qrels
```

Depois de alteradas as propriedades necessárias para habilitar o uso de **stopwords e stemming,** foram feitos todos os passos descritos nas alíneas anteriores (batchindexing, batchretrieve (para cada método) e por fim batchevaluate.

Desta forma, os resultados finais foram os seguintes:

```
C:\Users\paulo\Documents\terrier-project-5.4\bin>terrier.bat batchevaluate -f -j
20:26:14.510 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/BM25_1.res
Average Precision: 0.1459
20:26:14.814 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/BM25_8.res
Average Precision: 0.4000
20:26:14.993 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/BM25_d_3_t_10_4.res
Average Precision: 0.2159
20:26:15.181 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_0.res
Average Precision: 0.3605
20:26:15.358 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_6.res
Average Precision: 0.4033
20:26:15.542 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/DPH_d_3_t_10_3.res
Average Precision: 0.4045
20:26:15.698 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/TF_IDF_2.res
Average Precision: 0.3710
20:26:15.866 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/TF_IDF_7.res
Average Precision: 0.4063
20:26:16.116 [main] INFO  o.terrier.evaluation.AdhocEvaluation - Evaluating result file: C:\Users\paulo\Documents\terrier-project-5.4\var\results/TF_IDF_d_3_t_10_5.res
Average Precision: 0.4102
```

Legenda:

- BM25_1.res: método BM25 aplicado às queries, sem expansão e sem stopwords e stemming;
- BM25_d_3_t_10_4.res: método BM25 aplicado às queries, com expansão, mas sem stopwords e stemming;
- BM25_8.res: método BM25 aplicado às queries, sem expansão, mas com stopwords e stemming;
- DPH_0.res: método DPH aplicado às queries, sem expansão e sem stopwords e stemming;
- DPH_d_3_t_10_3.res: método DPH aplicado às queries, com expansão, mas sem stopwords e stemming;
- DPH_6.res: método DPH aplicado às queries, sem expansão, mas com stopwords e stemming;
- TF_IDF_2.res: método TF_IDF aplicado às queries, sem expansão e sem stopwords e stemming;
- TF_IDF_2_d_3_t_10_5.res: método TF_IDF_2 aplicado às queries, com expansão, mas sem stopwords e stemming;
- TF_IDF_2_7.res: método TF_IDF_2 aplicado às queries, sem expansão, mas com stopwords e stemming;

RESULTADOS:

	BM25	**DPH**	**TF_IDF**
Sem expansão e sem stopwords e stemming	0.1459	0.3605	0.3710
Sem expansão, mas com stopwords e stemming	0.4000	0.4033	0.4063

Através tabela, podemos concluir que o **uso de stopwords e stemming** melhora os resultados, isto é, as precisões com as stopwords e stemming são superiores aos valores obtidos sem estes.

Isto acontece neste caso em particular, mas não se pode generalizar, uma vez que a melhoria ou não do uso stopwords e stemming depende do tipo de queries utilizadas. O que tem que acontecer é que, se melhora num dos métodos, tem de melhorar os resultados para os restantes, e se a precisão diminuir para um dos métodos, então também deverá diminuir para os restantes. Tudo dependerá do tipo de queries.

Para finalizar, observa-se também que o que foi concluído anteriormente sem expansão e sem stopwords/stemming volta aqui a verificar-se, agora com estas opções. Ou seja, se anteriormente o método TF_IDF era o mais preciso, aqui comprova-se que, mesmo com o uso de stopwords e stemming, continua a ter um valor de precisão superior. Da mesma forma, o método BM25 continua a ser o que tem uma precisão mais baixa. No entanto, as diferenças são menos significativas.