universidade de aveiro

ASSIGNMENT 4 - RELATÓRIO

Evaluation methods for Relevance Feedback and thesaurus based query expansion

Recuperação de Informação

Curso: Engenharia de Computadores e Telemática

Departamento de Eletrónica, Telecomunicações e Informática

21 de dezembro de 2017

Professor:

Prof. Sérgio Matos

Autores:

Miguel Oliveira nº 72638

Tiago Henriques nº 73046

universidade de aveiro

Índice

1 – Introdução

O Assignment 4 tem como objetivo a aplicação dos métodos de avaliação para o *relevance feedback* e thesaurus com base na *query expansion* usando o corpus utilizado nas iterações anteriores. Assim, pretende-se implementar o método abordado nas aulas teóricas, Rocchio Relevance Feedback, que irá dividir-se em relevância explícita (onde são considerados os documentos relevantes e não relevantes) e relevância implícita (onde são apenas considerados os documentos relevantes). Ainda nesta iteração, pretende-se implementar a expansão da query com o recurso da biblioteca externa word2vec.
Por fim, para avaliar o desempenho do programa, aplicamos uma nova métrica, Normalized Discounted Cumulative Gain (NDCG) calculada para cada query e no fim é feita a média dos seus valores.
Serão ainda referidas as alterações efetuadas sobre as iterações anteriores, e por fim, serão expostos os resultados da execução do programa como o tempo e o total de memória utilizada.

2 - Diagrama de Classes

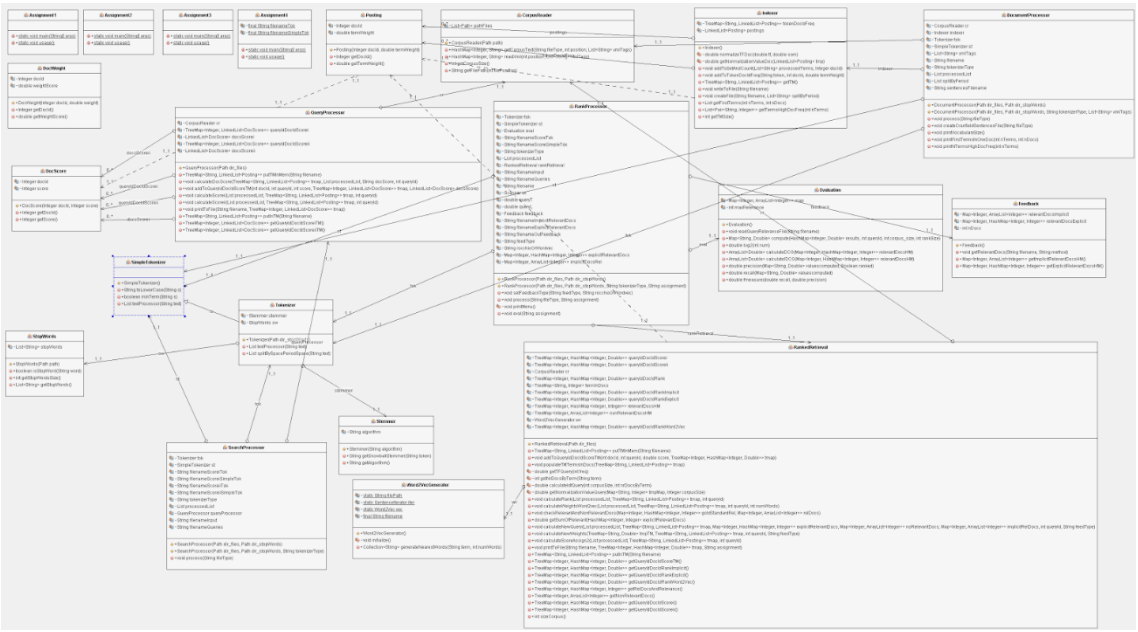


Figura 1 - Diagrama de Classes onde estão presentes as classes e os seus respetivos métodos

3 – Classes e Métodos

Nesta secção serão apenas descritas e explicadas as classes e os métodos associados que foram acrescentados ou alterados para fazer a iteração em questão.

3.1 – Assignment4 (main)

Classe principal e executável do projeto. Aqui, é instanciado o RankProcessor que iremos descrever mais abaixo.

3.2 – Feedback

Esta classe é instanciada na classe RankProcessor e foi adicionada ao projeto com o objetivo de processar os documentos relevantes e colocá-los na estrutura de dados adequada para depois ser possível efetuar o método de avaliação de "relevant feedback". Como foi dito, é necessário dividir o relevant feedback em dois tipos: implícito e explícito. No implícito estão a ser considerados os 10 primeiros documentos para cada query, sendo estes documentos considerados todos relevantes. Já para o modo explicito, na mesma são considerados os 10 primeiros documentos para cada query, mas aqui é necessário distinguir entre documentos relevantes e não relevantes. O método responsável por esta diferenciação é o getRelevantDocs(...) que vai ser explicado de seguida.

Métodos adicionados e a sua descrição:

- **getRelevantDocs(String filename, String method):**
  - Método responsável por diferenciar o tipo de relevância que vai ser usada como feedback, ou seja, caso o argumento de entrada "method" seja igual a "implicit" então será encaminhada o processamento para o método Implícito, caso seja igual a "explicit", então o método será o Explícito. É passado por argumento, a string "filename" que traduz o nome do ficheiro de onde vão ser lidos os documentos relevantes. Assim, para o método implícito, vai ser lido o ficheiro "fileout_docscoreranked_simple_3.txt" ou o "fileout_docscoreranked_tok_3.txt" que contém valor do score (tf-idf) associado a cada query e documento. Para cada query, serão então lidos os 10 primeiros documentos e neste caso todos estes documentos serão

considerados como relevantes, sendo posteriormente armazenada a queryId como key e uma ArrayList de docIds associados como os values da estrutura de dados definida como Map <Integer, ArrayList<Integer>>.
Para o método explícito, o ficheiro de onde vão ser lidos os documentos e a relevância associada é o documento "*gold standard",* já utilizado na terceira iteração. Como neste método os 10 primeiros documentos terão (numa fase mais tarde que vai ser explicada) de ser dividimos em relevantes ou não relevantes, é necessário guardar a relevância associada a cada um deles. Por esta razão, foi usada a uma Map<Integer, HashMap<Integer,Integer>>, onde a chave é o número da queryId e o valor é uma HashMap constituída pelo docId e a sua relevância.
Por fim, de notar que quando armazenamos a relevância associada ao documento, é guardado o valor (5 - relevância), de maneira a dar mais relevância aos documentos com relevância inferior no ficheiro *"gold standard*".

- **getImplicitRelevantDocsHM():**
  - Método que retorna a estrutura de dados associada ao método implícito, Map<Integer, ArrayList<Integer>> relevantDocsImplicit.
- **getExplicitRelevantDocsHM():**
  - Método que retorna a estrutura de dados associada ao método explícito, Map<Integer, HashMap<Integer,Integer>> relevantDocsExplicit.

4.3 – Evaluation

Esta classe é responsável por calcular as métricas de avaliação e eficiência de forma a poder inferir se os resultados foram ou não positivos e se o sistema é ou não eficiente. Para além das métricas anteriormente implementadas na iteração anterior, foi agora acrescentada um novo método de avaliação e eficiência, o Normalize Discounted Cumulative Gain (NDCG). Assim, nesta classe, irão ser apenas abordados os novos métodos implementados de modo a implementar a nova métrica, NDCG. De notar que o cálculo do valor médio desta métrica é realizado na classe RankProcessor, onde para cada query, é realizada a divisão do valor NDC pelo valor IDCG. Para esta métrica, apenas estamos a considerar os 10 primeiros documentos, ou seja, métrica para Rank 10. Neste caso, é adotado o mesmo processo utilizado na iteração anterior, de forma que o valor do NDCG será o valor médio de todas as queries.

Métodos e a sua descrição:

- **log2(int num):**
  - Método complementar que tem o propósito de calcular o logaritmo de base 2 de um número passado como argumento de entrada.
- **calculateDCG(Map<Integer, HashMap<Integer, Integer>> relevantDocsHM):**
  - Método responsável pelo cálculo do Discounted Cumulative Gain (DCG). É passado por argumento a estrutura de dados que contém os documentos

relevantes e a relevância associada para cada query. Este método começa por percorrer esta estrutura e recolher os values da Map, ou seja, os docIds e a sua relevância. Por cada documento percorrido, é incrementada uma variável de contagem iniciada em zero que irá permitir ver a posição do documento. Assim, exceto para a posição zero, é feito o somatório do valor da relevância sobre o logaritmo de base 2 da posição em questão do documento. Como o logaritmo de base 2 de 1 é um valor infinito, para este caso, o valor do DCG é apenas o valor da relevância. Assim, para cada query, é retornada uma ArrayList<Double> de valores de DCG.

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

- **calculateIDCG(Map<Integer, HashMap<Integer, Integer>> relevantDocsHM):**
  - Método responsável por calcular o valor de IDCG, onde tal como é feito para o DCG, são utilizados para cada queryId os docIds e a sua relevância. No entanto, agora as relevâncias serão armazenadas numa ArrayList<Integer> ordenada de maior para o menor (modo decrescente). De seguida, é percorrida esta ArrayList, e tal como na função anterior, a posição para cada relevância é importante e é feito o somatório de relevâncias a dividir pelo logaritmo de base 2 da posição. No fim, para cada query, é retornado uma ArrayList<Double> que contém os valores de IDCGs para essa query.

$$NDCG_n = \frac{DCG_n}{IDCG_n},$$

4.4 – Word2VecGenerator

Nesta classe, é implementado o processo de query expansion que irá ser aplicado a cada termo da query em questão. Esta classe irá ser instanciada na classe RankedRetrieval e tem o objetivo de retornar as 4 palavras mais semelhantes a cada termo da query de acordo com um método de aprendizagem e irão ser posteriormente adicionadas à nossa estrutura. Para o sistema puder prender de melhor forma e encontrar as palavras mais semelhantes era necessário um corpus de maior dimensão, então, usamos o ficheiro de treino definido pelo Professor e disponibilizado no elearning, “aerodynamics_sentences.txt.gz”.

Métodos e a sua descrição:

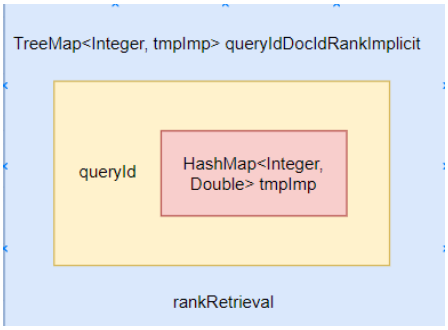
- **initialize():**
  - Método onde é implementado o processo de "deeplearning" pelo uso de Neural Network que processa o texto em wordvectors;
- **generateNearestWords(String term, int numWords):**
  - Método que retorna as 4 palavras mais similares do termo passado como argumento.

4.4 – RankedRetrieval

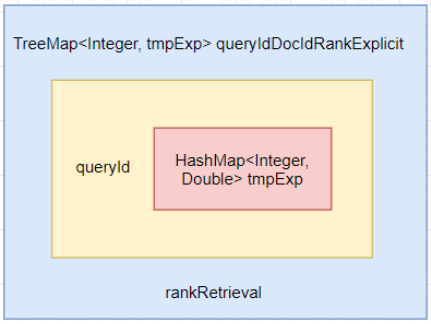
Estruturas de dados usadas:

As estruturas de dados pensadas e implementadas têm como objetivo principal o aumento do desempenho da pesquisa e de ranking. Portanto, consideramos que a melhor estrutura de dados a adotar seria:

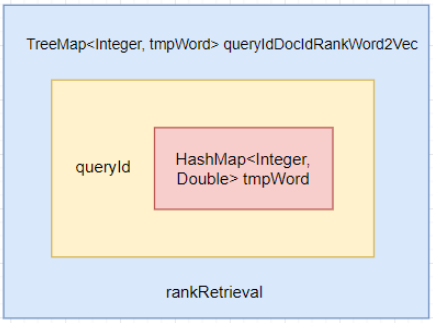
TreeMap<Integer, HashMap<Integer,Double>> queryIdDocIdRankImplicit: Este tipo de dados herda as características de uma TreeMap, em que a chave corresponde ao identificador da query e o valor corresponde a outra HashMap onde a chave é um identificador do documento (docId) e o valor é o respetivo score (wtd*wtq), quando é utilizado o modo implícito como feedback relevante.



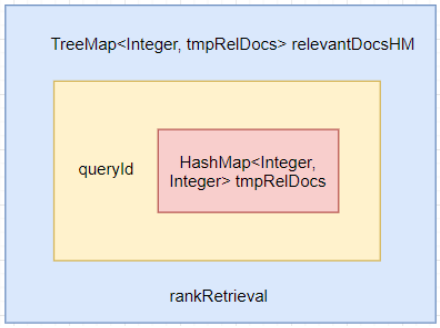
TreeMap<Integer, HashMap<Integer,Double>> queryIdDocIdRankExplicit: Este tipo de dados herda as características de uma TreeMap, em que a chave corresponde ao identificador da query e o valor corresponde a outra HashMap onde a chave é um identificador do documento (docId) e o valor é o respetivo score (wtd*wtq), quando é utilizado o modo explícito como feedback relevante.



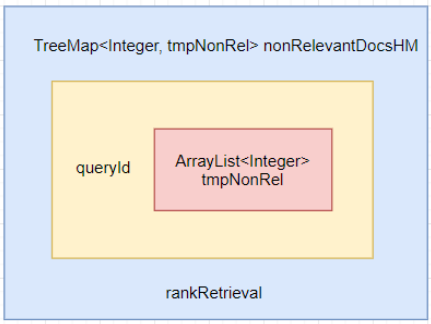
TreeMap<Integer, HashMap<Integer,Double>> queryIdDocIdRankWord2Vec: Este tipo de dados herda as características de uma TreeMap, em que a chave corresponde ao identificador da query e o valor corresponde a outra HashMap onde a chave é um identificador do documento (docId) e o valor é o respetivo score (wtd*wtq), quando é aplicada a técnica de Query Expansion.



TreeMap<Integer, HashMap<Integer, Integer>> relevantDocsHM: Este tipo de dados herda as características de uma TreeMap, em que a chave corresponde ao identificador da query e o valor corresponde a outra HashMap onde a chave é um identificador do documento (docId) e o valor é o valor de relevância desse documento relevante na query.



TreeMap<Integer, ArrayList<Integer>> nonRelevantDocsHM: Este tipo de dados herda as características de uma TreeMap, em que a chave corresponde ao identificador da query e o valor corresponde a uma ArrayList de inteiros que corresponde aos identificadores de documentos considerados não relevantes (docIds) para essa query.



Esta classe é instanciada na classe RankProcessor. Esta classe tem como objetivo implementar os métodos associados ao carregamento do Indexer em disco e ao cálculo do score, ou seja, é nesta classe que primeiramente são carregados os termos existentes no Indexer criado na iteração e ligeiramente adaptado nesta iteração para permitir a indexação por pesos (iteração 3). O score pode agora ser de três tipos (2 tipos de score da iteração 2 (score tipo 1 e score tipo 2) e o score dos pesos (doc+query) utilizado nesta iteração e na anterior). É também nesta classe que são definidos os métodos que permitem a implementação do algoritmo de Rocchio bem como a expansão de queries. Nesta classe são então implementados métodos auxiliares para o cálculo dos três tipos de scores e permite também, através da função printToFile(...), armazenar os resultados de cada tipo de scoring em disco através da escrita destes em ficheiros distintos.

Métodos e a sua descrição:

- **calculateWeightsWord2vec**(List processedList, TreeMap<String, LinkedList<Posting>> tmap, int queryId, int numWords)
    - Este método tem como propósito efetuar o cálculo do ranking para uma dada query aplicando a *query expansion* com o recurso da biblioteca word2vec, ou seja, este método é praticamente igual ao calculateRank(...) da iteração anterior, mas aqui são adicionados termos semelhantes à query para cada termo dessa mesma query. Portanto, após expandir a query, é utilizado o mesmo processo da iteração anterior para o cálculo dos pesos, ou seja, tf-idf.

- **checkRelevantAndNonRelevantDocs**(Map<Integer, HashMap<Integer, Integer>> goldStandardRel, Map<Integer, ArrayList<Integer>> relDocs)
    - Método auxiliar usado para o "*relevance feedback*" explícito para definir quais são os documentos relevantes e os não relevantes para cada query. Assim, o primeiro argumento deste método, refere-se é a estrutura de dados resultante da leitura do ficheiro "*gold standard*" e o segundo parâmetro, refere-se à estrutura de dados que contém os valores lidos do ficheiro "fileout_docscoreranked_simple_3.txt" ou

"fileout_docscoreranked_tok_3.txt" (consoante o tokenizer usado). Caso os documentos da estrutura de dados relDocs estejam presentes na estrutura de dados goldStandardRel, então esses documentos são considerados relevantes, caso contrário, são considerados não relevantes. Por fim, para cada query, são armazenados os docIds dos documentos e a sua relevância (para o caso dos relevantes) na estrutura de dados adequada, TreeMap<Integer, HashMap<Integer, Integer>> relevantDocsH, para os documentos relevantes e para os não relevantes são armazenados os docsIds dos documentos não relevantes para cada query na estrutura TreeMap<Integer, ArrayList<Integer>> nonRelevantDocsHM.

- **getSumOfRelevant**(HashMap<Integer, Integer> explicitRelevantDocs):
  - Método que retorna a soma das relevâncias dos documentos relevantes de modo explícito presentes na estrutura de dados passada como argumento.

- **calculateNewQuery**(List processedList, TreeMap<String, LinkedList<Posting>> tmap,Map<Integer, HashMap<Integer, Integer>> explicitRelevantDocs, Map<Integer,ArrayList<Integer>> noRelevantDocs, Map<Integer, ArrayList<Integer>> implicitRelDocs, int queryId, String feedType):
  - Método chamado quando é pretendida a implementação do algoritmo de rocchio. Este método tem um funcionamento bastante similar ao do método calculateRank(...) especificado na iteração anterior, mas aqui, dependendo do tipo de feedback pretendido (explícito ou implícito), é calculado o novo score da query. Para o modo explícito são considerados o feedback positivo (documentos relevantes) e o feedback negativo (documentos não relevantes). Para o modo implícito, são é apenas considerado o feedback positivo (documentos relevantes). De notar, que os valores de alpha, betha e miu considerados foram aqueles que se encontram nos apontamentos teóricos, ou seja, foi considerado o valor de 1 para o alpha, o valor de 0.75 para o betha e o valor de 0.25 para o miu. No final deste método, é chamado outro método, calculateNewWeights(...) que irá ser explicado de seguida.

- **calculateNewWeights**(TreeMap<String, Double> tmpTM, TreeMap<String, LinkedList<Posting>> tmap, int queryId, String feedType):
  - Método responsável por associar os pesos da nova query à estrutura de dados correspondente, ou seja, caso esteja a ser implementado o algoritmo de rocchio através do modo explícito, é chamada a função addToQueryIdDocIdScoreTM(...) (já explicada em iterações anteriores) com a estrutura de dados do modo explícito (queryIdDocIdRankExplicit) a ser passada como argumento. Caso esteja a ser implementado o algoritmo de rocchio através do modo implícito, a estrutura de dados do modo implícito (queryIdDocIdRankImplicit) é passada como argumento para esta função.

- **getQueryIdDocIdRankImplicit**()
  - Método que retorna a TreeMap queryIdDocIdRankImplicit.

- **getQueryIdDocIdRankExplicit**()
  - Método que retorna a TreeMap queryIdDocIdRankExplicit.

- **getQueryIdDocIdRankWord2Vec**()
  - Método que retorna a TreeMap queryIdDocIdRankWord2Vec.

- **getRelDocsAndRelevance**()
  - Método que retorna a TreeMap relevantDocsHM.

- **getNonRelevantDocs**()
  - Método que retorna a TreeMap nonRelevantDocsHM.

4.5 – RankProcessor

Esta classe é instanciada no programa que é executado nesta iteração (Assignment4) quando se pretende o fluxo relativo ao cálculo de pesquisa por ranking e posterior cálculo das métricas de avaliação. Esta classe pode ter dois fluxos neste assignment, ou é definido um fluxo onde é implementado o Algoritmo de Rocchio ou então é implementado um fluxo onde é aplicado o modo de extensão de queries com recurso à biblioteca "word2vec". Tal como em iterações anteriores, esta classe instancia a classe RankedRetrieval que é responsável por inicialmente carregar o indexer previamente gerado em memória, e posteriormente permitir uma pesquisa ordenada por ranking, ou seja, processa os termos de cada query e aplica o cálculo do score para cada documento tendo por base a query. Aqui, também é instanciada a classe responsável pela classificação dos resultados, a classe Evaluation, bem como a classe Feedback, onde são definidos os documentos considerados relevantes para o processo de "relevance feedback" usandos os dois métodos existentes (Explícito e Implícito) de modo a aplicar o algoritmo de Rocchio. Durante a execução do programa, é definido um menu de interação com o utilizador para que este possa escolher que tipo de métrica deseja ser apresentada (introdução da métrica, Normalize Discounted Cumulative Gain (NDCG)).

Métodos e a sua descrição:

- **setFeedbackType(String feedType, String rocchioOrWordvec):**
  - Método responsável por definir que processo usar, Rocchio ou word2vec através da string rocchioOrWordvec e o tipo de método que irá ser usado no relevance feedback, explícito ou implícito, pela definição da string feedType.

- **process(String fileType, String assignment):**
  - Este método para além de ser responsável pelo processo do cálculo do score, onde começa por carregar o indexer previamente gerado em memória, e posteriormente processa os termos de cada query e aplica o cálculo do score para cada documento tendo por base a query e o tokenizer aplicado, agora referente a esta iteração, começa por determinar quais os documentos relevantes para cada método de *relevance feedback* e coloca as estruturas usadas em memória. Depois dependendo do processo escolhido aplica o algoritmo de Rocchio ou a query expansion usando o word2vec.

- **printMenu():**
  - Método responsável por imprimir o menu das opções que vão do 1 ao 10 (de acordo com as métricas especificadas).

- **eval(String assignment):**
  - Neste método é processado o fluxo do processo de classificação dos resultados, ou seja, é instanciada a classe Evaluation e invocados os seus métodos para as diversas métricas. Assim, o método começa por carregar a estrutura TreeMap<Integer, HashMap<Integer,Double>> resultante do cálculo do score. Nesta iteração apenas foi acrescentado o cálculo da métrica NDCG onde, como foi dito anteriormente, é calculado pela divisão do DCG com o IDCG. O resultado final desta métrica será a média do valor de todas as queries percorridas.

5 - Diagrama geral de fluxos

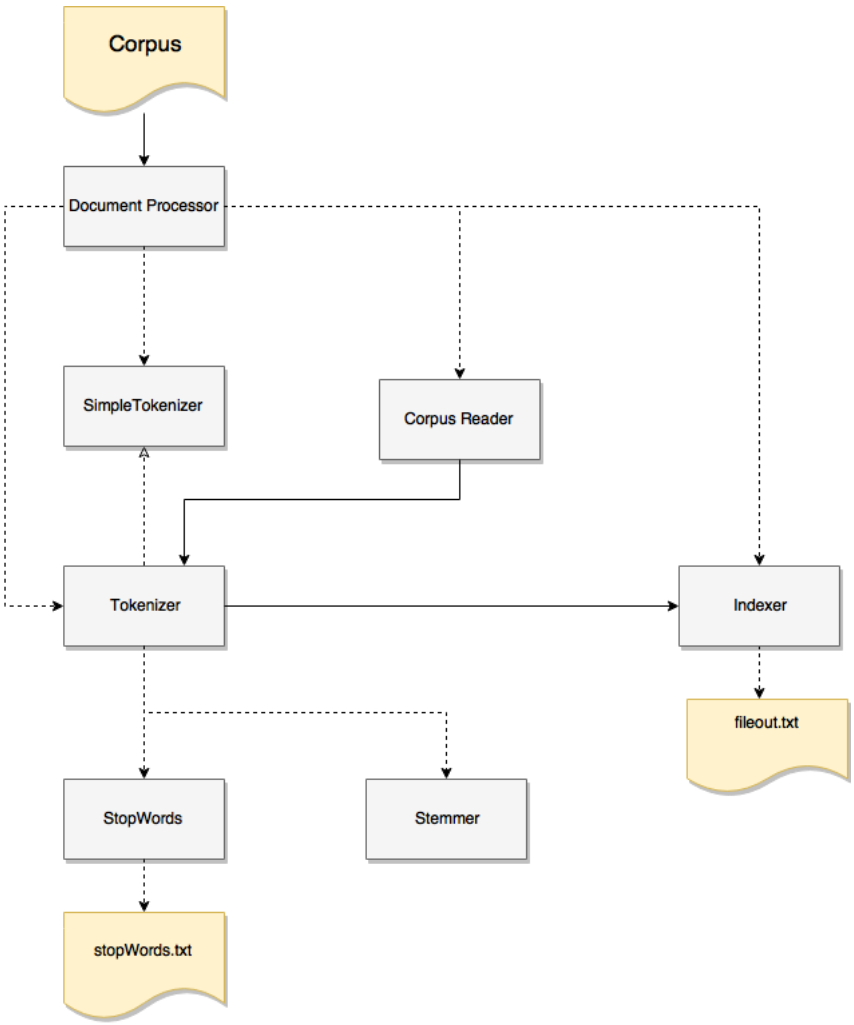


Figura 4 - Imagem que representa o fluxo de execução do programa através do processo de indexação por pesos

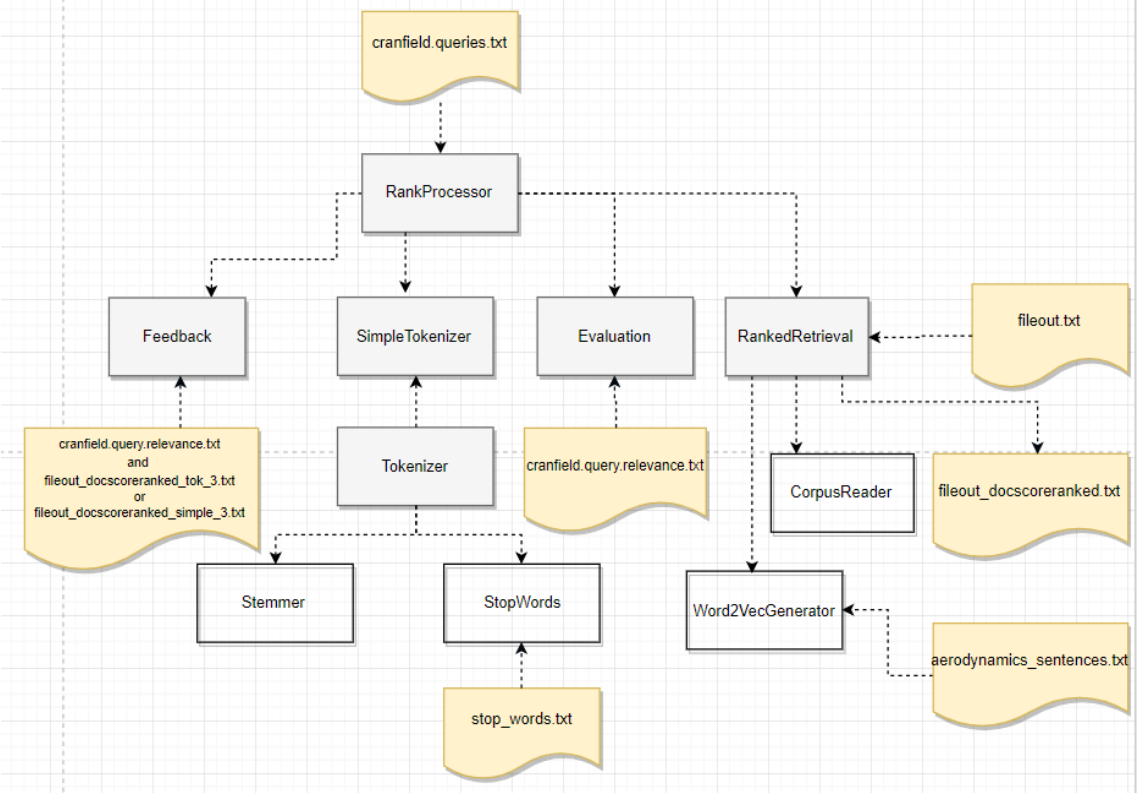


Figura 5 - Imagem que representa o fluxo de execução do programa através do processo de pesquisa por ranking

A figura anterior mostra-nos de forma esquemática o fluxo de dados no nosso projeto. Caso seja desejado a indexação dos termos, o fluxo do programa será o esquematizado em (1), tendo por base o DocumentProcessor devidamente detalhado na iteração 1 e 3. Caso seja desejado a pesquisa por ranking, o fluxo do programa será o esquematizado em (2), encontrando-se detalhado na descrição da classe RankProcessor, aplicando agora o algoritmo de Rocchio e a expansão de queries.

7 - Estrutura do código

A estrutura do nosso projeto *Maven* encontra-se organizada da seguinte forma:

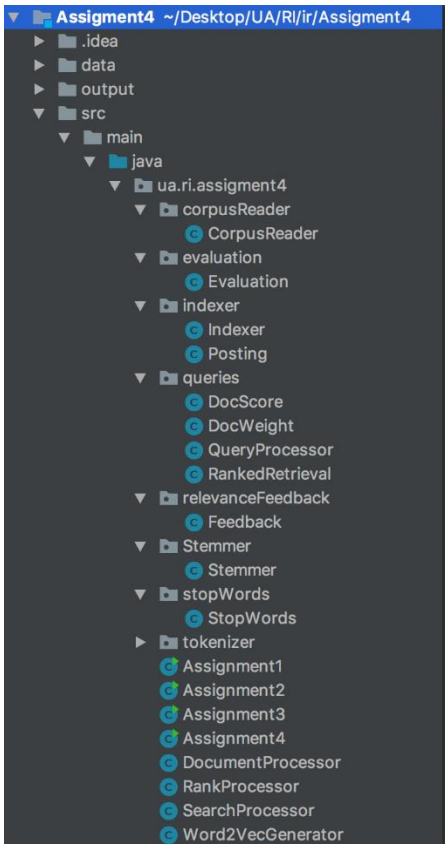


Figura 12 - Descrição da estrutura do nosso projeto Maven

No diretório raiz do nosso projeto Maven é possível encontrar três diretórios principais:

• **data:** Este diretório, tal como tem sido habitual ao longo de todas as iterações, está dividido em subdiretórios. No subdiretório "cranfield", encontra-se a coleção de corpus utilizados. No subdiretório "queries", encontra-se o ficheiro das queries considerado para pesquisa, "cranfield.queries.txt". Ainda neste subdiretório, encontra-se também o ficheiro "cranfield.query.relevance.txt", relevante para o processo de cálculo das métricas de avaliação.

Na raiz do diretório, existem 3 ficheiros distintos:

- **stopWords.txt:** contém todas as stopwords consideradas.

- **cranfield_sentences.txt**: Inicialmente era o ficheiro utilizado para treinar o modelo do wordvec para efetuar "query expansion". De momento, não está a ser considerado pois já foi fornecido no elearning um outro ficheiro já com as palavras processadas que devemos usar como ficheiro para treino, "aerodynamics_sentences.txt".
- **aerodynamics_sentences.txt:** Ficheiro final onde o algoritmo do word2vec é treinado para encontrar as palavras mais semelhantes

• **output:** Este diretório vai conter o(s) ficheiro(s) gerado(s) no processo de indexação. Este(s) ficheiro(s) terá(ão) o nome "fileout_simple.txt" caso tenha sido utilizado o simple tokenizer e o nome "fileout_tok.txt" no caso de ter sido utilizado o tokenizer complexo, tal como nas iterações anteriores. Os ficheiros gerados na terceira iteração foram também colocados neste diretório. Na 3ª iteração foi também gerado um ficheiro para cada tipo de tokenizer utilizado. O ficheiro tem o nome "fileout_docscoreranked_tok_3.txt" no caso de ser lido o ficheiro indexado pelo tokenizer complexo. No caso de ser lido o ficheiro indexado pelo tokenizer simples, o ficheiro tem o nome "fileout_docscoreranked_simple_3.txt". Relativamente às melhorias do Assignment2 realizadas na última iteração, os ficheiros previamente gerados no processo de indexação dessa iteração, têm o nome "fileout_simple_assign2.txt" e "fileout_tok_assign2.txt". Os ficheiros gerados no processo de pesquisa têm o nome "fileout_docscoreranked_tok_2_i.txt" para o tipo de score 1, e o nome "fileout_docscoreranked_tok_2_ii.txt", para o tipo de score 2.

Relativamente aos ficheiros gerados nesta iteração, no processo de "relevance feedback" dependendo do método escolhido os ficheiros gerados são "implicitRelevanceFeedback.txt", para o método implícito e "explicitRelevanceFeedback.txt" para o explícito. Caso seja utilizado o processo de wordvec ("query expansion") é gerado um ficheiro com o nome "word2vecRelevanceFeedback.txt".

• **Dependencies:** neste diretório estão colocados as bibliotecas compiladas necessárias para este projeto. Para este caso foi usado o jar do Snowball (Relativo à primeira iteração do trabalho) e a biblioteca externa utilizada para aplicar a expansão das queries (Word2Vec - DeepLearning4Jj).

8- Bibliotecas Externas

The Porter stemming algorithm: http://snowball.tartarus.org/.

De referir, que esta biblioteca externa apenas foi utilizada na primeira iteração do projeto.

Word2Vec - DeepLearning4Jj: https://deeplearning4j.org/word2vec

Biblioteca externa utilizada para aplicar a expansão das queries. Dependências desta biblioteca foram devidamente adicionadas ao ficheiro pom.

9- Execução

Para executar o nosso projeto é necessário a seguinte abordagem:

- compilar e executar o nosso projeto recorrendo ao Netbeans ou outro IDE semelhante

- podem ser passados, por argumento, o tipo de tokenizer que se pretende correr como 1º argumento. O 2º parâmetro diz respeito ao tipo de fluxo requerido, indexer ou ranker (mesmo funcionamento da iteração 3). Como 3º parâmetro, é passado o método a usar no "relevance feedback", ou seja, se se pretende usar o modo explicito ou o modo implícito. Como 4º argumento, é ainda possível determinar o processo pretendido, se é escolhido fazer avaliação do relevance feedback ou se é escolhido proceder com o método de expansão de queries. Modo de uso: **java Assignment4 <tokenizer or simple> <indexer or ranker> <explicit or implicit> <rocchio or wordvec>**.

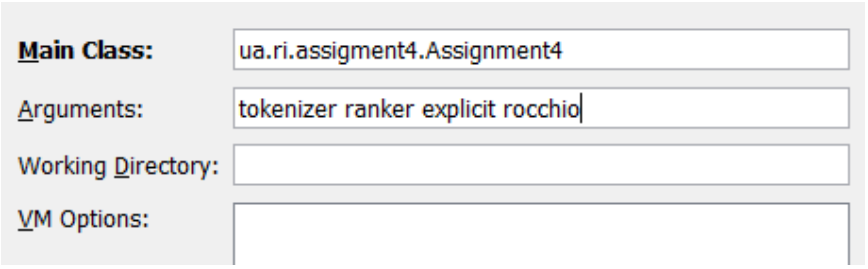
No entanto, por defeito, é considerado o tipo de Tokenizer complexo, a opção de ranker, o relevance feedback explícito e a implementação do algoritmo de Rocchio, por isso, apenas é necessário correr o comando**: java Assignment4**.

De notar que o tipo de tokenizer apenas aceita a string <tokenizer> que corresponde ao Tokenizer complexo ou a string <simple> que corresponde ao Simple Tokenizer. Caso seja introduzida uma outra string, nenhum tipo de tokenizer será reconhecido e será apresentada uma mensagem no terminal com o formato que deve ser utilizado para correr o programa.

O método de relevance feedback apenas aceita a string <explicit> que corresponde ao modo explícito ou a string <implicit> que corresponde ao modo implícito. O último parâmetro passado como argumento apenas aceita a string <rocchio> que corresponde à implementação do algoritmo de rocchio ou a string <wordvec> que corresponde ao processo de query expansion.

Caso seja introduzida uma outra string, nenhum tipo de fluxo de programa será reconhecido e será apresentada uma mensagem no terminal com o formato que deve ser utilizado para correr o programa.

Exemplo de modo de execução do programa:



10- Resultados

Para obtermos os tempos que do processo de indexação e respetiva geração e escrita em ficheiro, foi utilizado o seguinte hardware:

• **Processador**: Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz

• **Memória**: 8.00 GB

• **Sistema Operativo**: Windows 10, 64bits

Para o corpus fornecido sample cranfield.zip e cranfield.queries.txt, foram obtidos os seguintes tempos:

Simulação	Tempo de execução (s)
1	29.548
2	29.282
3	29.866
4	29.429

O tempo médio de execução foi: 29,531 segundos

De notar que o tempo de execução está a contemplar o período de aprendizagem seguindo o modelo word2vec do DeepLearning4j, daí resultar em tempos bastantes mais elevados do que em iterações anteriores.

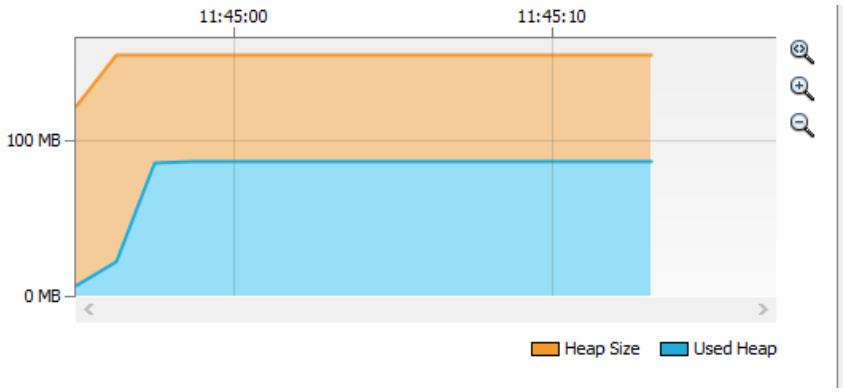


Figura 13 - Descrição de medidas de desempenho: CPU, Memória e Garbage Collection

11- Resultados das métricas de classificação e de eficiência

Nesta secção iremos apresentar os resultados as métricas de avaliação. Como foi dito anteriormente, para efetuar os cálculos estamos apenas a considerar os 10 primeiros documentos, portanto, será efetuado NDCG Rank 10.
Ainda estamos a apresentar as métricas de classificação e de eficiência definidas na terceira iteração de maneira a podermos comparar os resultados obtidos com a iteração passada.

Usando o Tokenizer Complexo:

Relevance Feedback Type	**NDCG (%)**
Explícito	93%
Implícito	73%

Métricas de Classificação - Explicit Feedback	
Mean Precision	0.008311
Mean Recall	0.959918
Mean F-measure	1,785054

Mean Average Precision	0,019216
Mean Precision at Rank 10	0,230000
Mean Reciprocal Rank	0.032485

Métricas de Eficiência	
Query throughput (queries/s)	1168
Median query latency (s)	856114

Métricas de Classificação - Implicit Feedback	
Mean Precision	0.008443
Mean Recall	0.959600
Mean F-measure	1.813068
Mean Average Precision	0.019182
Mean Precision at Rank 10	0,230000
Mean Reciprocal Rank	0.032720

Métricas de Eficiência	
Query throughput (queries/s)	1833
Median query latency (s)	545571

Usando o Tokenizer Simples:

Relevance Feedback Type	NDCG %
Explícito	93%
Implícito	73%

Métricas de Classificação - Explicit Feedback	
Mean Precision	0.006315
Mean Recall	0.97893
Mean F-measure	1.397429
Mean Average Precision	0.014599
Mean Precision at Rank 10	0.170000
Mean Reciprocal Rank	0.028980

Métricas de Classificação - Implicit Feedback	
Mean Precision	0.008443
Mean Recall	0.959600
Mean F-measure	1.813068
Mean Average Precision	0.019182
Mean Precision at Rank 10	0,230000
Mean Reciprocal Rank	0.032720

Métricas de Eficiência	
Query throughput (queries/s)	1275
Median query latency (s)	784329

Word2Vec:

Usando o Tokenizer Complexo:

Relevance Feedback Type	NDCG %
Explícito	93%
Implícito	73%

Métricas de Classificação - Explicit Feedback	
Mean Precision	0,007929
Mean Recall	0,961075
Mean F-measure	1,728488
Mean Average Precision	0,017554
Mean Precision at Rank 10	0,200000
Mean Reciprocal Rank	0,031660

Métricas de Eficiência	
Query throughput (queries/s)	19
Median query latency (s)	53115534

Métricas de Classificação - Implicit Feedback	
Mean Precision	0,007957
Mean Recall	0,960899
Mean F-measure	1,727830
Mean Average Precision	0,017738
Mean Precision at Rank 10	0,200000
Mean Reciprocal Rank	0,031690

Métricas de Eficiência	
Query throughput (queries/s)	18
Median query latency (s)	56677068

12 - Repositório de desenvolvimento

Para o desenvolvimento deste trabalho foi utilizado a plataforma CodeUA de forma a potenciar o trabalho de equipa. O repositório é o seguinte: http://code.ua.pt/projects/ir/repository.

13 - Conclusão

No final da quarta e última iteração podemos dizer que ao longo de todas as iterações fomos adquirindo os conhecimentos obtidos na cadeira e aplicando-os na prática de forma a elaborar um trabalho modular e incrementalmente progessivo de modo a atingir os resultados esperados no trabalho final.

Em relação a esta iteração, a avaliação de "relevance feedback" através do uso do algoritmo de Rocchio e a query expansion através do word2vec foram conceitos adquiridos nas aulas teóricas tendo sido colocados em prática de modo a obter resultados que possam ser comparados com os das iterações anteriores. De notar que para o cálculo do NDCG decidimos apenas considerar os 10 primeiros documentos pedidos tanto para fazer o feedback implícito como explícito.

Podemos ainda concluir que a utilização do modelo word2vec piora ligeiramente os resultados obtidos em termos de performance e tempo relacionado com o throughput e latência de queries, em comparação com os que obtivemos nas iterações anteriores.

Por fim, achamos que tivemos um percurso progressivo e incremental tendo vindo a melhorar algumas pequenas implementações de modo a melhorar o trabalho anterior e atingir resultados mais satisfatórios e mais eficientes.