

Assignment 1

Recuperação de Informação

Trabalho realizado por:

- Lucas Barros nmec : 83895

- Pedro Cavadas nmec : 85090

Introdução

O principal objetivo deste trabalho é o de criar um programa que processa vários documentos de texto e analise os campos PMID e TI. De seguida são extraídos tokens, ou seja, palavras chave de maneira a auxiliar a implementação de um mecanismo de busca numa fase posterior.

Este trabalho também ajuda a perceber as tarefas que estão por detrás dos melhores motores de busca da atualidade como o Google, sempre que se escreve uma Query e se pressiona ENTER.

A linguagem de Programação adotada foi Python 3 por ser simples, e por desempenhar as funções de processamento de texto (strings) com grande eficiência.

Conceitos Teóricos

Tokenization

Tokenization é o processo de converter uma sequência de caracteres de entrada em tokens de saída. Este processo tem normalmente um conjunto de subprocessos/regras associados que vão depender de Tokenizer para Tokenizer. Neste projeto, utilizamos 2 Tokenizers para fazer comparações.

Indexer

Indexer é uma estrutura de dados que permite que objetos estejam guardados como listas. Neste caso, o nosso Indexer vai ter o formato “token : doc\_id : [index1, index2]” etc…

Isto vai permitir fazer procuras na nossa estrutura de dados de forma rápida e estruturada.

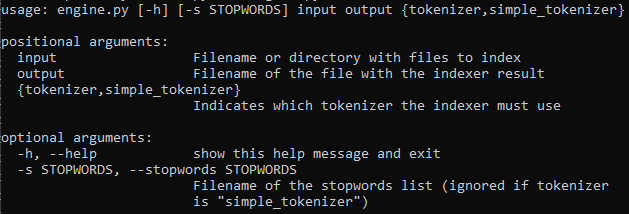
Passos para executar o Projeto

Diagrama de Classes

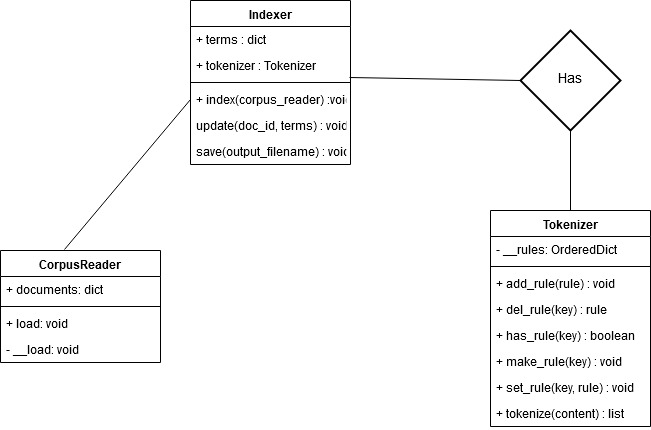
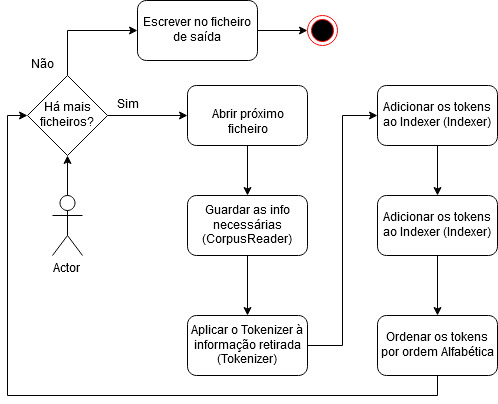


Diagrama de Workflow



Como se pode observar pelos diagramas, o texto original (documentos) são processados pela classe “CorpusReader”. Esta classe possui apenas um único método, o construtor em que lê o ficheiro e guarda num dicionário “documents” (pmid : TI) o ID do documento (PMID) e como valor, o TI correspondente numa string.

De seguida, os TI’s vão ser processados pela classe Tokenizer que vai aplicar uma série de regras. Estão implementados 2 Tokenizers, o “simple tokenizer” e o “tokenizer”. O “simple tokenizer” substitui todos os caracteres não alfabéticos por espaços, substitui todos os caracteres pelo minúsculo correspondente. A divisão dos tokens é efetuada por espaço e são ignorados todos os tokens com menos de 3 caracteres.

Já o “tokenizer” substitui todos os caracteres pelo minúsculo correspondente, substitui apenas certos sinais de pontuação por espaços, sendo eles: “.”, “,”, “-”, “!”, “?”, “;”, “\*”, “/”, “=”, “(“, “)”, “[“, “]”, “:”, ‘“‘, “‘“, “\”, “\n”, com a exceção do “.” quando este é precedido de um único caratere alfabético seguido e no máximo precedido de um caratere alfabético seguido, transformando assim uma sequência deste tipo de pontos num único token, por exemplo, “u.s.a.” ou “u.s.a” são ambos convertidos para “usa”. Além do ponto, também o “-” não é substituído quando é antecedido e precedido por uma palavra, por exemplo, “algodão-doce”. Além disto filtra todas as palavras com menos de 2 carateres, além de utilizar uma *stopword* *list* para filtrar certas palavras especificas, bem como aplica stemming a todas as palavras. Os tokens, tal como anteriormente são separados utilizando os espaços.

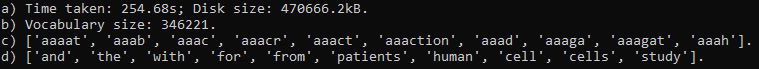
De seguida os tokens são indexados pela classe Indexer que possui um dicionário “terms” que tem a seguinte estrutura:

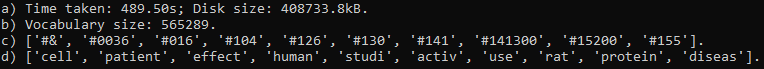
{token : {doc\_id1 : [index1, index2], doc\_id2 : [index3]}}

em que token como o próprio nome indica é o token, o doc\_id é o PMID do documento, e os index são as posições no texto em que o token apareceu. Esta classe possui o método “update(doc\_id, terms)” que sempre que é processado um documento, indexa os tokens desse documento.



Análise de Resultados

Simple Tokenizer

Tokenizer

O que se pode observar nas imagens acima são:

1. Tempo demorado em segundos e espaço ocupado em disco;
2. Tamanho do vocabulário;
3. 10 tokens que apenas aparecem num único documento;
4. 10 tokens mais frequentes;

Ambos os *tokenizer*s levaram tempos diferentes em diferentes máquinas, apesar disso os tempos rondaram sempre à volta de 260 segundos para o *simple tokenizer* e de 500 segundos para o *tokenizer*.

Podemos concluir também que as StopWords têm muita influência no tamanho em disco ocupado pelo Indexer pois o Tokenizer apesar de ter um vocabulário muito maior ocupa menos espaço em disco por causa das StopWords que lhe foram retiradas que por aparecerem em muitos documentos, ocupam muito espaço no disco.



Conclusão

Com este trabalho aprendemos mais sobre processamento de texto e também sobre as várias tarefas desempenhadas pelos motores de busca como a Google.

Podemos concluir também que os resultados obtidos são muito satisfatórios e fazem sentido de acordo com as regras apontadas a cada Tokenizer como explicado na Análise de Resultados.

Referências

<https://medium.com/@datamonsters/text-processing-in-python-step-tools-and-examples-bf025f872908>

<https://www.stackoverflow.com>

<https://medium.com/towards-artificial-intelligence/text-mining-in-python-steps-and-examples-78b3f8fd913b>