TESTANDO O TANTIVY

NATASHA ROCHA E GUILHERME FRANCO

8 de Maio de 2019

CONTEÚDO

RESUMO

O objetivo do trabalho é testar um sistema de recuperação de informação e, para tanto, escolhemos testar o Tantivy e sua interface de linha de comando. Tantivy é um sistema morderno de indexação e busca de documentos escrito em Rust, uma linguagem de programação multiparadigma com uma comunidade muito forte. Rust segue invicta como a linguagem de programação mais amada por desenvolvedores desde 2016, de acordo com o *Stack Overflow Developer Survey*. [1][2][3][4] Mostraremos como indexar documentos e realizar consultas com o Tantivy, assim como todas as etapas de pré-processamento que realizamos com os dados.

1 O QUE É O TANTIVY?

Tantivy é uma biblioteca em Rust para a criação de sistemas de recuperação de informação. Ele não é um sistema pronto para ser usado, mas sim uma ferramenta para se construir um. Nesse sentido ele se assemelha mais a sistemas como o *Apache Lucene* do que *Elasticsearch* ou *Apache Solr*.

Algumas das funcionalidades da biblioteca estão descritas a seguir:

- Busca em texto completo: as consultas são realizadas na totalidade do texto do documento, ao invés de se basear em metadados ou seções específicas como títulos e referências;
- Tokenização configurável: stemming disponível para 17 línguas latinas, além de pacotes desenvolvidos por terceiros para Chinês e Japonês;
- Rapidez: testes de benchmark mostram que o Tantivy é em geral mais veloz que Lucene;
- Indexação multithread: é possível indexar toda a biblioteca em Inglês do Wikipedia em menos de três minutos.
- Tempo de startup de menos que 10 ms;
- Rankeamento BM25;
- Buscas com operadores lógicos e de frases específicas (buscar por uma frase inteira ao invés de por palavras-chave);

Apesar do Tantivy não ser uma solução pronta, ele possui uma interface de linha de comando que facilita a criação de um sistema de recuperação de informação com o mesmo, chamada tantivy-cli. Essa é a ferramenta que estaremos testando.

2 INSTALAÇÃO DA FERRAMENTA

Para usar o tantivy-cli você precisa do Rustup, que instala todas as ferramentas necessárias para o desenvolvimento em Rust. Em uma máquina com sistema operacional tipo Unix (Linux ou macOS, por exemplo), a forma mais fácil de realizar a instalação é digitando o seguinte comando no terminal:

```
> curl https://sh.rustup.rs -sSf | sh
```

Existe um instalador para Windows, que pode ser baixado pela seguinte página. Estaremos instalando o tantivy-cli com o Cargo, o gerenciador de pacotes para Rust que foi instalado pelo Rustup.

```
> cargo install tantivy-cli
```

Caso apareça algum erro durante a instalação altere a versão do Rust para a 1.32.0.

```
> rustup default 1.32.0
```

3 PRÉ-PROCESSAMENTO

3.1 Pré-processamento dos documentos

O Tantivy exige que os documentos estejam em um formato específico. Eles precisam estar em um único arquivo, onde cada linha desse arquivo é um objeto JSON representando um documento.

Os documentos originais estavam em formato SGML, espalhados entre vários arquivos. Cada documento estava contido em uma tag <DOC> e possuía alguns outros atributos, como número do documento, data ou texto.

```
<DOC>
   <DOCNO>FSP950214-010</DOCNO>
   <DOCID>FSP950214-010</DOCID>
   <DATE>950214</DATE>
   <CATEGORY>BRASIL</CATEGORY>
   <TEXT>
      ...
   </TEXT>
</DOC>
```

Exemplo de documento da coleção

Consideramos apenas o número do documento e o texto contido dentro do mesmo e descartamos o resto, pois não seriam úteis para a nossa aplicação.

Apesar do Tantivy permitir a configuração do seu próprio tokenizador, aproveitamos que os dados teriam que ser pré-processados para realizar a etapa de stemming e remoção de stopwords a priori. Usamos como referência o corpus de stopwords em Português do NLTK (Natural Language Toolkit), uma biblioteca em Python para processamento de linguagem. Para realizar o stemming usamos a implementação do RSLP stemmer dessa mesma bilbioteca.

3.2 Pré-processamento das consultas

Para as consultas consideramos apenas o texto dentro de PT-Title. Removemos manualmente os acentos e usamos as mesmas ferramentas para stemming e remoção de stopwords que foram utilizadas no pré-processamento dos documentos. Além disso, fizemos as seguintes alterações nas consultas:

- Mudança de "Actividades" para "Atividades" na consulta "Actividades da ETA em França". Em todos os documentos disponibilizados só há uma ocorrência de "Actividades" , portanto fizemos essa alteração;
- Remoção do termo "passados" na consulta "Filmes passados na Escócia". Observando os documentos retornados quando "passados" estava incluído, não achamos que o termo tem relevância significativa;
- Mudança do termo "extinção" para "extinta" (feito o stem, *extinc* e *extint*, respectivamente). Utilizando o termo "extinção" eram retornados poucos documentos, a maioria sem relevância para a consulta. Observamos que a busca feita com "extint" retornou documentos que se encaixam melhor com o objetivo da consulta. Consideramos o termo "vias"como stopword também, já que não parecia relevante para a consulta.

4 INDEXAÇÃO DOS DOCUMENTOS

Para indexar documentos com o Tantivy eles precisam estar em JSON. Todos os documentos precisam seguir uma mesma estrutura, ou seja, possuir os mesmos campos. Nenhum a mais e nenhum a menos. O primeiro passo para a indexação é definir qual será a estrutura dos documentos, ou schema.

Para criar um schema pela interface de linha de comando do Tantivy você precisa criar uma pasta para o seu index (/index, por exemplo).

```
> mkdir index
```

Após criar a pasta, adicione um schema à ela com o seguinte comando:

```
> tantivy new -i index
```

Um assistente irá te guiar durante a criação do schema.

```
Creating new index
Let's define its schema!



New field name ? docno
Text or unsigned 32-bit integer (T/I) ? T
Should the field be stored (Y/N) ? Y
Should the field be indexed (Y/N) ? N
Add another field (Y/N) ? Y



New field name ? text
Text or unsigned 32-bit integer (T/I) ? T
Should the field be stored (Y/N) ? N
Should the field be indexed (Y/N) ? Y
Should the field be tokenized (Y/N) ? Y
Should the term frequencies (per doc) be in the index (Y/N) ? Y
Should the term positions (per doc) be in the index (Y/N) ? N
Add another field (Y/N) ? N

[
{
   "name": "docno",
   "type": "text",
   "options": {
      "indexing": null,
      "stored": true
   }
},
{
   "name": "text",
   "type": "text",
   "options": {
      "indexing": {
         "record": "freq",
         "tokenizer": "en_stem"
      },
      "stored": false
   }
},
]
```

Ao final da execução será criado um arquivo *meta.json* com o schema, que você poderá alterar para customizar a indexação. No nosso caso não queríamos usar o *tokenizer en_stem* que foi definido automaticamente, já que a etapa de *stemming* já havia sido realizada durante o pré-processamento dos documentos. Além disso, esse stemmer é otimizado para textos em Inglês, e não Português. Mudamos então o *tokenizer* para o *default*, que apenas coloca as palavras em minúsculo e tokeniza usando como separadores pontuação e espaços em branco. Com o schema pronto e os documentos no formato correto, podemos então finalmente indexar a coleção.

```
> cat collection.json | tantivy index -i ./index
```

Comando para indexar a coleção

5 REALIZAÇÃO DE CONSULTAS

Com os documentos já indexados você pode realizar suas consultas de três formas diferentes. Você pode fazer pesquisas usando a cli, criando um servidor de busca ou então configurando seu próprio buscador em Rust. Optamos pelo servidor de busca.

5.1 Criando um servidor de busca local

A CLI do Tantivy permite que você sirva seu sistema de busca através de um servidor. Ele pode ser ativado com o seguinte comando:

```
> tantivy serve -i index
```

Por padrão ele será servido na porta 3000. Depois de inicializar o servidor as consultas são feitas através de uma API RESTful, que retorna os resultados da consulta como um JSON.

```
http://localhost:3000/api/?q=michel+temer&nhits=100
```

Exemplo de consulta

Essa busca, por exemplo, seria lida como *michel OR temer*. Podemos ignorar documentos com certas palavras colocando um traço na frente das mesmas, ou um sinal de + (%2B) para retornar apenas documentos que contenham pelo menos uma ocorrência da palavra. Aspas também permitem procurar por frases exatas.

5.2 Nossas consultas

Realizamos dois conjuntos de consultas: primeiro buscamos por documentos que não necessariamente continham todos os termos, realizando uma busca simples (por exemplo, boicot consum). No segundo teste usamos o símbolo + para buscar por documentos possuíam todos os termos da consulta (por exempo, +boicot +consum). Para o segundo caso algumas consultas acabaram retornando o documentos, então resolvemos isso removendo a obrigatoriedade de algumas palavras. Na consulta "produc glob opi", por exemplo, pesquisamos por "produc glob +opi".

Verificamos que o segundo conjunto retornou um número muito menor de documentos, porém mais relevantes. O primeiro retornou um número enorme de documentos, sendo a maioria não relevante para a consulta, pois observamos documentos retornados que não necessariamente possuíam todos os termos.

REFERÊNCIAS

[1] Stack overflow developer survey 2016. https://stackoverflow.com/insights/survey/2016#technology-most-loved-dreaded-and-wanted.

[2] Stack overflow developer survey 2017. https://stackoverflow.com/insights/survey/2017#most-loved-dreaded-and-wanted.

[3] Stack overflow developer survey 2018. https://insights.stackoverflow.com/survey/2018/#most-loved-dreaded-and-wanted.

[4] Stack overflow developer survey 2019. https://insights.stackoverflow.com/survey/2019#technology-_-most-loved-dreaded-and-wanted-languages.