Universidade Federal do Rio de Janeiro
Centro de Ciências Matemáticas e da Natureza
Instituto de Matemática
Departamento de Ciência da Computação

Trabalho 1:
Teste de um Sistema de Recuperação de Informação Apache Solr

Alunos:
Gustavo Monteiro
Leticia Freire

maio/2019

Apache Solr

Apache Solr é um banco de dados não relacional que usa o Apache Lucene que é um software de indexação e recuperação de documentos. O Solr é um servidor que oferece integração via uma API REST e um dashboard.

A versão mais moderna é o Solr 8 que tem como dependência o Java na versão 8. Ele funciona sem a necessidade de instalação. Ao baixar o compacto no site e o extrair, o servidor já pode ser comandado pelo script *bin/solr*.

Para o trabalho, criamos um script em shell script, que foi testado em um ambiente Ubuntu 18. O script garante que o ambiente possua o Java 8, baixa o Solr, levanta o servidor e já cria uma *collection* com o nome *'noticias'* e configura o campo *'text'*. Por haver instalações, é necessário que o script seja executado com usuário *root*. O script utilizado pode ser visto no **Apêndice 1**.

No Solr, existe a *collection* que é um conjunto de documentos. Cada *collection* possui um *schema* que define a estrutura dos documentos, e como eles serão indexados e recuperados. O caso da *'noticias'* o schema é *'data driven',* o que quer dizer que o schema se molda aos dados. Para o trabalho, é favorável que o campo *'text'* fosse entendido como um texto em português (type *'text_pt'*) por questão de pré processamento que será detalhado mais à frente.

Para inserir os documentos na coleção, construímos um script em Python, utilizando a biblioteca PySolr. O script passa os documentos para uma estrutura de dicionário e envia utilizando o comando `solr.add(documents)`, adicionamos todos os documentos no Solr. O script construído pode ser consultado no **Apêndice 2.**

Ao final, além de utilizar o framework que o Solr oferece para realizarmos as consultas - o framework será mais explorado nos tópicos seguintes - também utilizamos Python e a biblioteca PySolr para isto. Definimos cada uma das consultas que seria realizada e utilizando o comando `solr.search(query, rows=100, fl="* score")`, retornamos até 100 documentos para a consulta *query*. O script pode ser consultado no **Apêndice 3.**

Funcionalidades

O sistema oferece um framework muito amigável para o usuário. Nele, é possível realizar operações de consulta , inserção e deleção de arquivos. Na tela inicial do Solr, ele fornece informações sobre suas especificações, como sua versão e as variáveis de ambiente que ele utiliza. Também traz informações sobre o sistema que ele está utilizando, a quantidade de memória que está sendo utilizada.

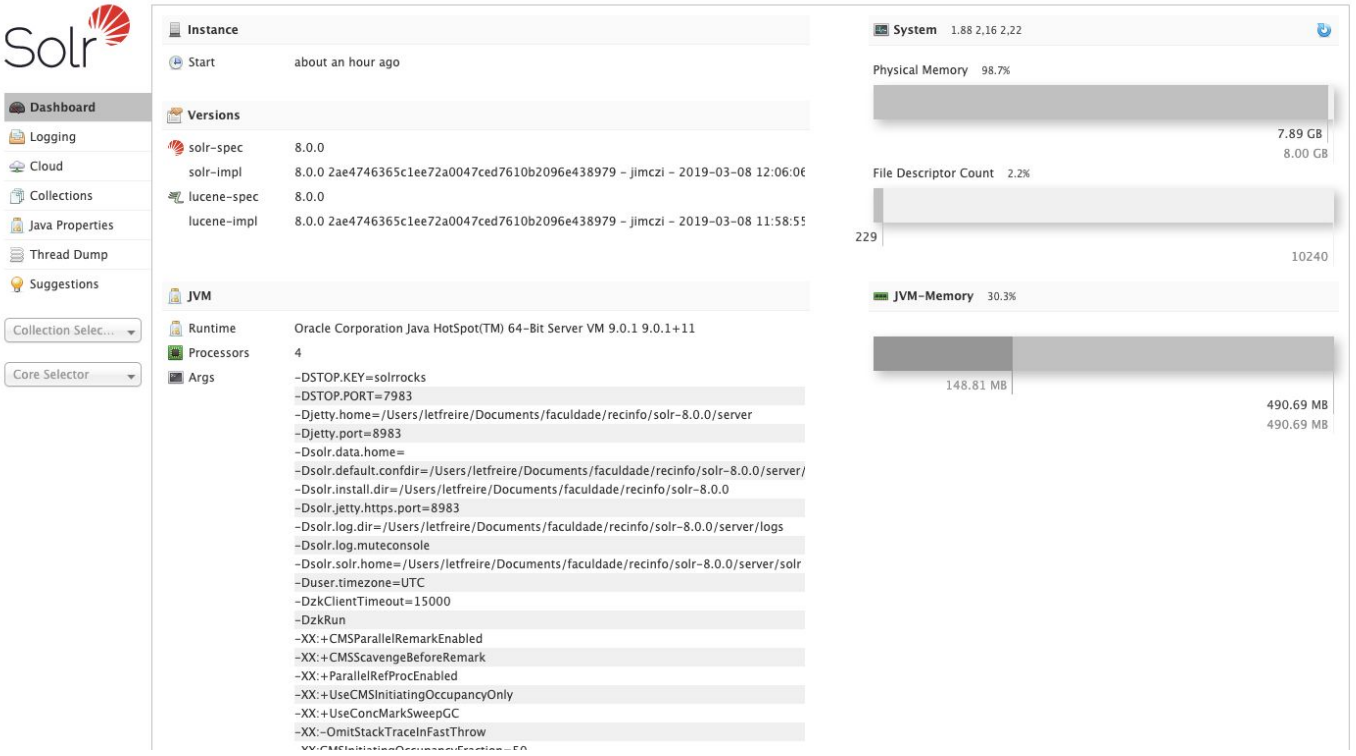

Figura 1: Tela inicial

No Solr, no seu lado esquerdo, é apresentado um Menu com opções sobre coisas que o usuário pode fazer com o solr de modo simples.

O Solr é uma plataforma que integra um banco de dados NoSQL. Desta forma, ele possui coleções e cada coleção possui documentos, como já dito anteriormente. Seria o equivalente, em um banco SQL, a uma tabela e suas tuplas, respectivamente.

Uma das opções presentes no menu é acessar informações e documentos presentes na coleção. Na figura abaixo, nós estamos acessando a coleção chamada *noticias*. Como visto na imagem, há muitas coisas a se fazer na coleção, sendo uma delas a *query*.

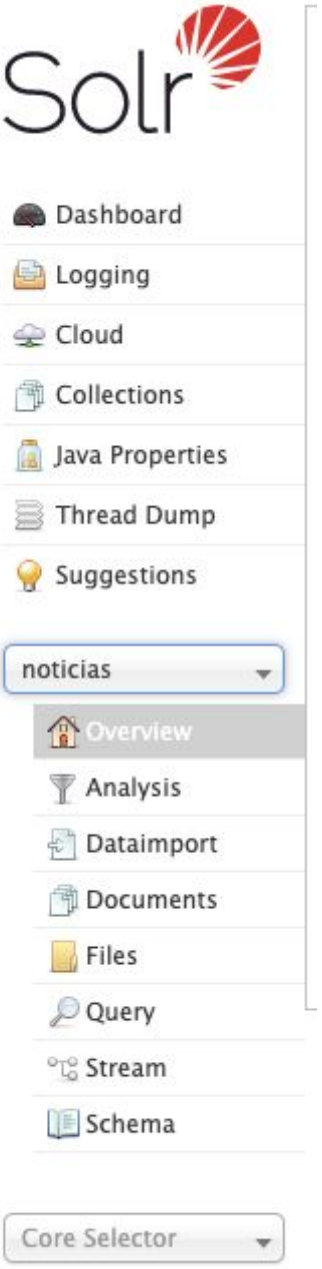

Figura 2: Menu

Na opção *query* é possível consultar os documentos presentes na coleção.

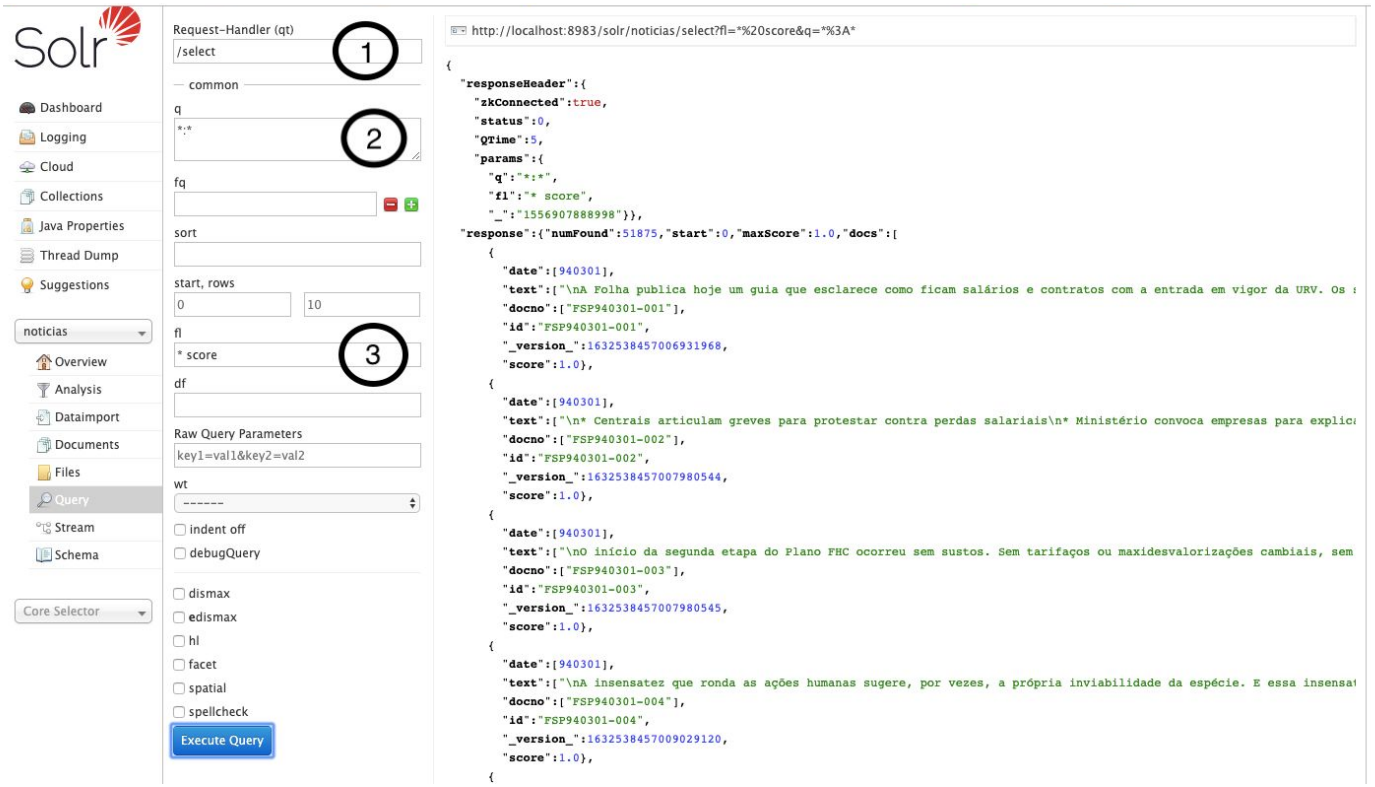

Figura 3: Tela de consulta aos documentos

Nesta tela, estão enumeradas algumas funções importantes presentes no framework:

Número	Nome da função	Função
1	Request-handler (qt)	Especifica que tipo de consulta será feita aos documentos [1]
2	q	A consulta que será feita aos documentos. Na figura 3, foi feita uma consulta por todos os documentos (*:*) [1]
3	fl	Define os campos a serem apresentados nos resultados da consulta. No caso acima, pedimos que retornasse todos os campos presentes nos documentos e o campo *score* de cada documento (* score) [1]

Tabela 1: Campos importantes no framework Solr

Seguindo o menu, há outro campo importante chamado *Files*. Ali estão presentes arquivos de configuração da coleção.[2]

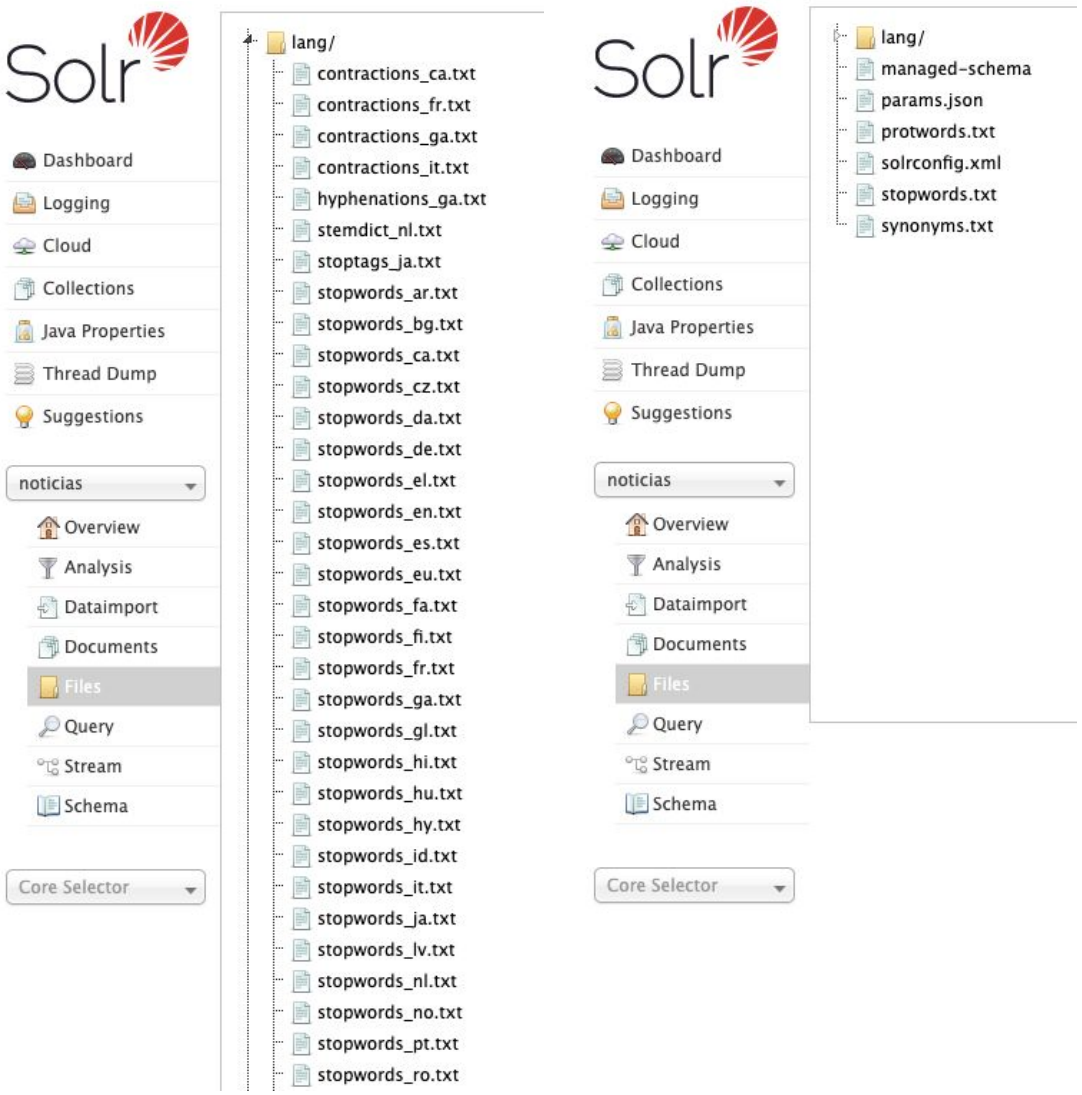

Figura 4: Arquivos de configuração da coleção

O arquivo *solrconfig.xml* define o comportamento do sistema ao indexar documentos. Os outros documentos serão configurados dependendo de como é definido este arquivo e a parte do *Schema*. O *Schema* define os tipos de dados que estarão presentes nos documentos da coleção, os campos em que os documentos serão divididos e outros campos dinâmicos que devem ser gerados de acordo com os documentos recebidos. [2]

Na parte de *Schema* também está definido o modelo que está sendo utilizado para a recuperação dos documentos. No Solr, está definido o modelo BM25, com os parâmetros k=1.2 e b=0.75.

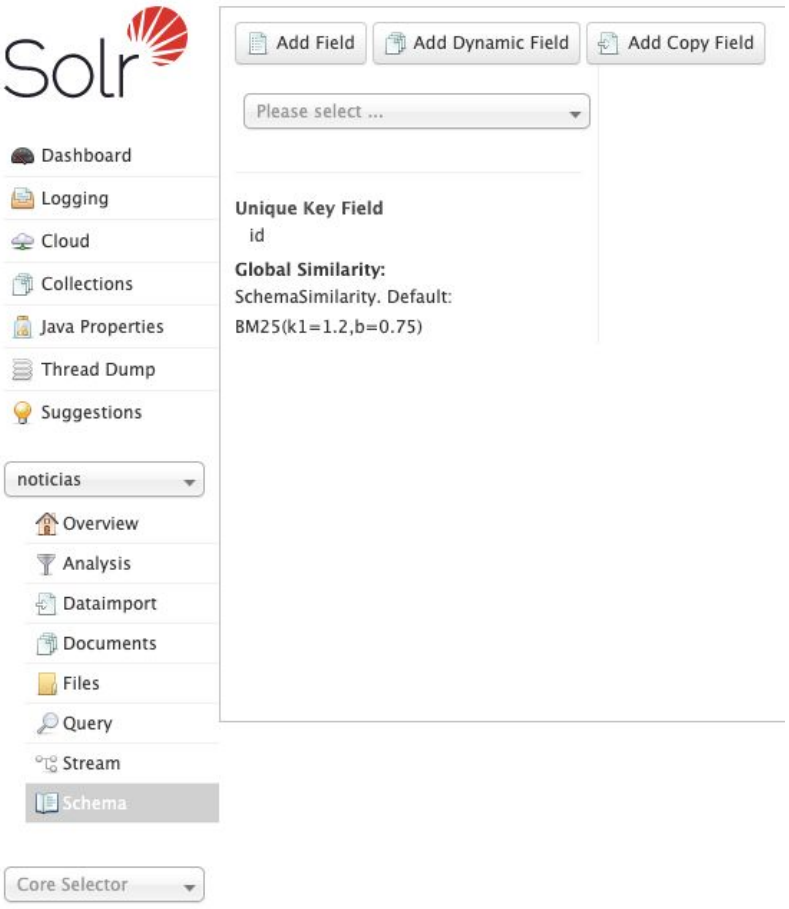

Figura 5: Definição do modelo

Pré-processamento

O Solr guarda os documentos como um conjunto de chaves-valor. Porém o valor é tipado, e todo o pré-processamento está atrelado ao tipo do valor.

```
<fieldType name="text_pt" class="solr.TextField" positionIncrementGap="100">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_pt.txt" format="snowball" />
      <filter class="solr.PortugueseLightStemFilterFactory"/>
      <!-- less aggressive: <filter class="solr.PortugueseMinimalStemFilterFactory"/> -->
      <!-- more aggressive: <filter class="solr.SnowballPorterFilterFactory" language="Portuguese"/> -->
      <!-- most aggressive: <filter class="solr.PortugueseStemFilterFactory"/> -->
    </analyzer>
  </fieldType>
```

Figura 6: Definição do tipo.

Para o trabalho usamos o tipo '*text_pt*' que já vem filtros de LowerCase, remoçao de stopwords e processamento de stem. Todos implementados pelo próprio solr.

No caso da *stemming*, o solr, oferece 4 formas de processamento possíveis. Eles se diferem em o quão agressivo a Stemização pode ser. Os mais agressivos possuem mais overstemming e os mais leves possuem mais understemming. Optamos pela sugestão do solr em usar o segundo mais leve.

No caso dos stopwords, o solr também já vem com uma lista para todas as linguagens suportadas que esta em *$SOLRHOME/server/solr/configsets/noticias_configs/conf/lang/stopwords_pt.txt*.

Experimentos

Experimento 1: consulta padrão

Para este trabalho, alimentamos a plataforma com os documentos fornecidos por CHAVE [3]. Para isto, construímos um script Python que lia os arquivos em formato .sgml e inseria na coleção *notícias* por meio de uma biblioteca chamada PySolr.

Após a inserção de todos os documentos necessários, começamos a entender como eram feitas as consultas. Por meio do framework que o Solr diponibiliza, inserimos diversar queries no campo *q*. A primeira delas foi a consulta padrão ***:*.** Esta consulta retorna todos os documentos presentes na coleção.

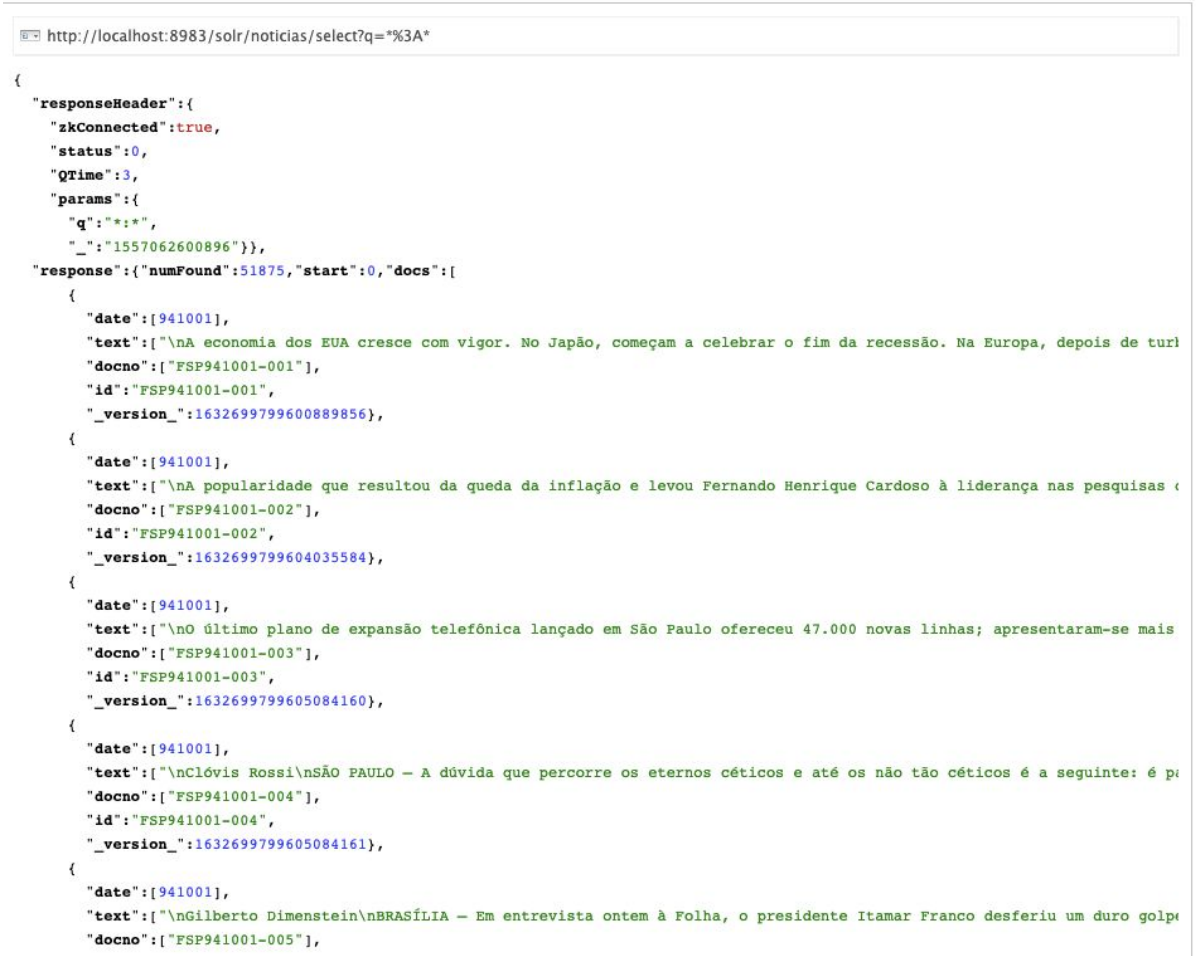

Figura 7: Exemplo da consulta padrão

Experimento 2: consulta por campo específico

Após isso, começamos a consultar utilizando um campo específico que estava presente nos documentos. Uma das consultas utilizadas foi por meio do campo *date*. Nos documentos que estamos analisando, existe um campo em que informa qual dia, mês e ano aquela notícia foi publicada. Quando feita a consulta **date:"941001",** indicando que estamos procurando notícias do dia 01/01/1994, obtemos os seguintes resultados:

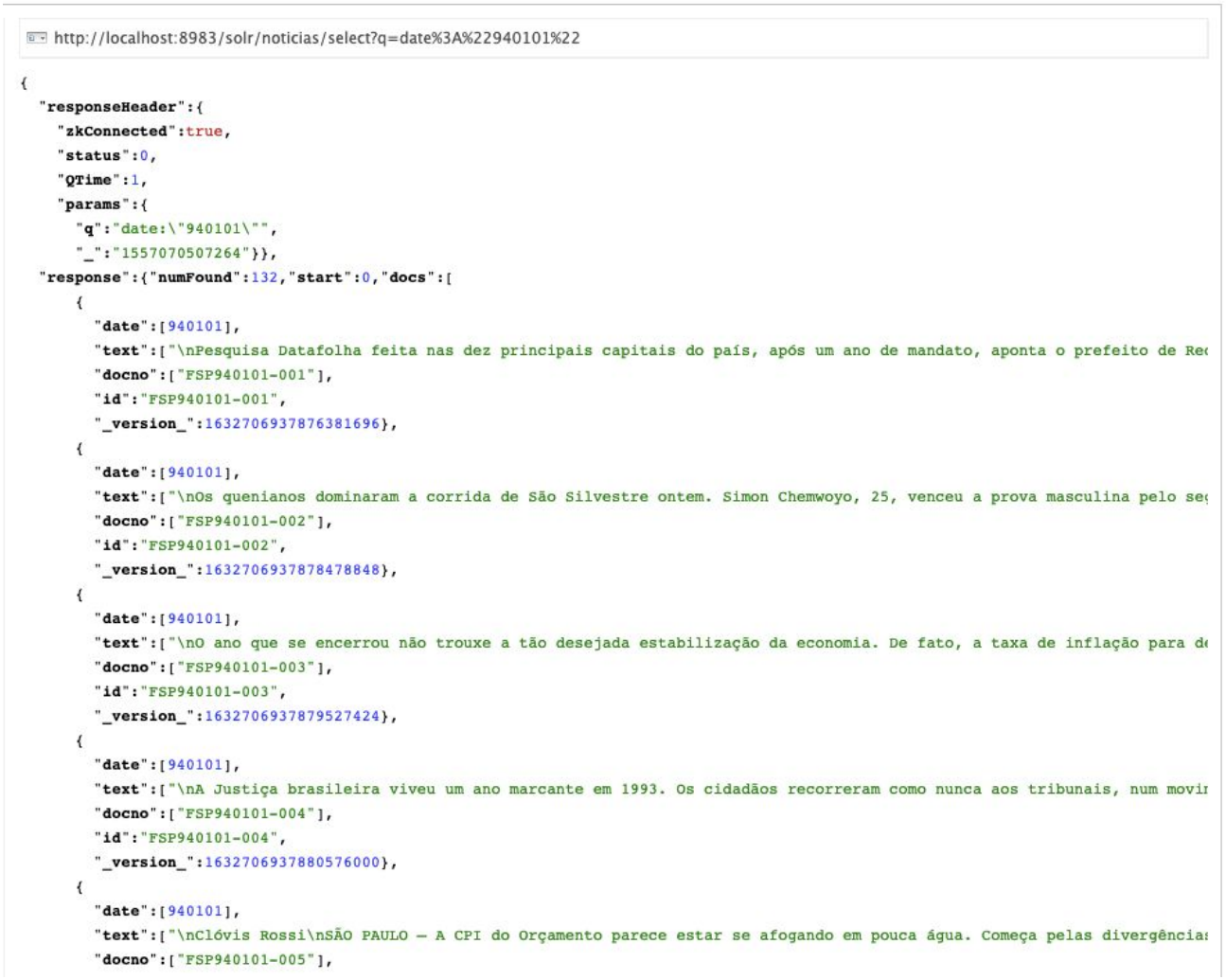

Figura 8: Exemplo de consulta por meio do campo date

Experimento 3: consulta utilizando o campo *fl*

Observando os resultados das consultas acima, vemos que são apresentados os campos dos documentos mas não é apresentado o *score* que é fornecido pelo modelo BM25. Para que ele seja apresentado nos resultados também, além da query, é necessário modificar o campo *fl* apresentado pelo framework. Então, refazendo a consulta utilizando o campo *date* e modificando o fl para *** score**, obtemos os seguintes resultados:

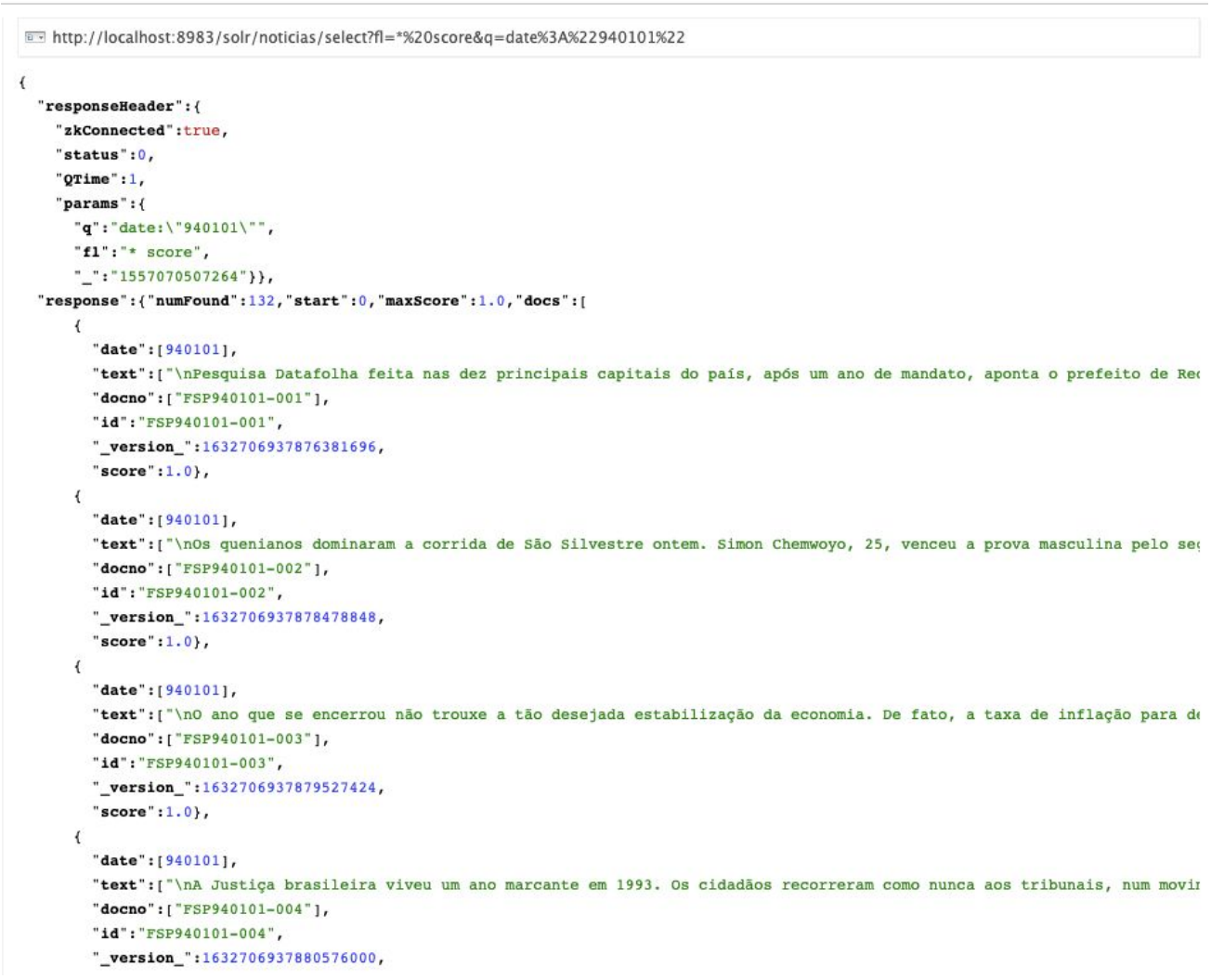

Figura 9: Exemplo de consulta modificando o campo fl

Como podemos observar, os resultados são os mesmos, mas, na figura 8, além dos campos presentes nos documentos também temos o campo *score*.

Experimento 4: consulta utilizando operadores lógicos

O Solr possui muitas ferramentas para criar consultas complexas. Uma delas são os operadores lógicos. [5] A operação OR é uma operação padrão. Exemplo: **text:(crise energética)** retornará documentos que ou tem o termo ‘crise’ ou ‘energética'.
A operação AND pode ser escrita **text:(crise AND energética)** ou **text:(crise && energética)** ou **text:(+crise +energética)** . Todos os exemplos geram a mesma resposta.
O sinal ‘+’ quer dizer que a palavra deve estar presente na consulta. A combinação pode ser feita com o uso dos parênteses. Exemplo: **text:((crise AND energética) (falta AND energia)).**

Podemos também usar a operação NOT. Exemplo **text:(terrorista AND França ! Espanha)** retorna apenas documentos que tenham os termos ‘terrorista’ e ‘França’ porém que não tenha o termo ‘Espanha’.

Outra ferramenta interessante e muito usada é a pesquisa difusa. Ao adicionar um símbolo '~' ao final da palavra, a resposta também irá conter documentos com termos semelhantes aquela palavra. Exemplo: **text:(França~)** também retornará documentos que contenham a palavra "francês" ou "francesas" entre outras.

Experimento 5: consultas do trabalho

Para este trabalho, foram feitas 10 descrições sobre quais documentos deveriam ser apresentados ao serem feitas as consultas referentes a estes pedidos. Através de tentativa erro, as consultas que nós consideramos que se encaixam melhor nas descrições são as seguintes:

1. text:(+boicote consumidores~ ! político~)
2. text:(terrorista~ +ETA +França~)
3. text:(filmes documentários Escócia~ gravação~)
4. text:(+resíduos industriais (métodos remoção))
5. text:(+desemprego Europa~ números~ taxa~ índice~)
6. text:('100º aniversário' centenário '100 anos')
7. text:(+espécies +extinção proteger~ Europa~ animal~)
8. text:(+greve greve~ causa~ objetivos~ motivo~)
9. text:(+ópio produção global~ mundial~ cultivo papoilas)
10. text:(+energia 'crise energética' energética~ crise combustível~ causa~)

Tabela 2: Consultas feitas no conjunto de documentos CHAVE

Reprodução do ambiente:

Para reproduzir o ambiente que usamos. Em uma máquina com Ubuntu 18:

- Executar '*sudo ./mysolr.sh'* . Esse script irá instalar o java 8 e subir o servidor Solr na porta 8983 com as configurações que usamos.
- Para executar *upload.py* é precisso das bibliotecas *pandas, pysolr* e *bs4*. Caso não tenha instalado é só executar '*sudo ./python_libs.sh*'.
- Executar "*python3 upload.py*" que os arquivos do diretório "*colecao_teste*" será inserido no solr.
- Acesse http://localhost:8983/solr/# ou http://<IP do servidor solr>:8983/solr/# para acessar o dashboard.

O script "*consulta.py*" foi feito apenas para fazer as consultas do trabalho e gerar um arquivo com os resultados.

Referências

[1] https://lucene.apache.org/solr/guide/6_6/query-screen.html
[2] https://lucene.apache.org/solr/guide/6_6/files-screen.html#files-screen
[3] https://www.linguateca.pt/CHAVE/
[4] https://lucene.apache.org/solr/guide/7_6/language-analysis.html#portuguese
[5] https://lucene.apache.org/solr/guide/6_6/the-standard-query-parser.html#TheStandardQueryParser-BooleanOperatorsSupportedbytheStandardQueryParser

Apêndice 1

```
#!/bin/bash
if [ "$EUID" -ne 0 ]; then
    echo "Favor rodar como root"
    echo "sudo ./mysolr.sh"
    exit
fi

install_java()
{
    apt-get update -y
    apt-get install default-jdk -y
}


if type -p java &> /dev/null; then
    version=$(java -version 2>&1 | awk -F '"' '/version/ {print
$2}')
    if [[ "$version" < "1.8" ]]
    then
        echo Instalando java 8
        install_java
    fi
else
    echo Instalando java
    install_java
fi

if [ ! -d "./solr-8.0.0" ]; then
    echo "Baixando Apache Solr"
    curl
"http://ftp.unicamp.br/pub/apache/lucene/solr/8.0.0/solr-8.0.0.tg
z" --output ./solr-8.0.0.tgz
    tar -xvzf solr-8.0.0.tgz
    rm solr-8.0.0.tgz
```

```
fi

cd ./solr-8.0.0
if ! bin/solr status &> /dev/null; then
    echo Subindo solr na porta 8983
    bin/solr start -c -p 8983 -force
fi

if [ ! -d "./server/solr/configsets/noticias_configs" ]; then
    echo Criando Configs para a colection noticias
    cp -r server/solr/configsets/_default
server/solr/configsets/noticias_configs
fi


if ! curl
http://localhost:8983/solr/admin/collections?action=LIST 2>
/dev/null | grep noticias &> /dev/null ; then
    echo criando a collections
    bin/solr create_collection -c noticias -d
server/solr/configsets/noticias_configs -n noticias_configs
-force
    curl -X POST -H 'Content-type:application/json' --data-binary
'{
    "add-field":{
        "name":"text",
        "type":"text_pt",
        "stored":true,
        "indexed":true,
        "uninvertible":true }
    }' http://localhost:8983/solr/noticias/schema
fi



echo Solr no ar!
```

Apêndice 1: Scrip para subir o Solr

Apêndice 2

```
from bs4 import BeautifulSoup
import pandas as pd
from calendar import monthrange
import pysolr
```

```
def read_file(path):
     file_doc = open(path,'r')
     return file_doc


def get_all_tags(file_docs, except_tegs=None):
    soup = BeautifulSoup(file_docs, 'html.parser')
    all_tags_set = set([tag.name for tag in soup.find_all()])
    all_tags = list(all_tags_set)
    if except_tegs:
        for teg in except_tegs:
            all_tags.remove(teg)
    return all_tags



def transform_df(file_docs, all_tags):
     soup = BeautifulSoup(file_docs, 'html.parser')
     dict_teste = {}
     for tag in all_tags:
         dict_teste[tag] = [doc.text for doc in
soup.find_all(tag)]
     return pd.DataFrame(dict_teste)


def connect_solr(host, collection, port=8983):
     path = 'http://{}:{}/solr/{}'.format(host, port,
collection)
     return pysolr.Solr(path)


def dict_docs(docs_dataframe):
     all_docs = []
     for index in range(0, len(docs_dataframe)):
          doc = docs_dataframe.ix[index]
          all_docs.append(doc)
     return all_docs

def add_solr(solr_client, all_docs_list):
     solr_client.add(all_docs_list)
     solr_client.commit()

solr_client = connect_solr('localhost', 'noticias')

#pegando do ano de 94 e 95
years = [94, 95]
for year in years:
     print("INSERINDO ANO {}".format(year))
     for month in range(1, 13):
          print("INSERINDO MES {}".format(month))
```

```
            year_complete = year + 1900
            days = monthrange(year_complete, month)[1]

            list_docs = []
            for day in range(1, days+1):
                  print("INSERINDO DIA {}".format(day))
                  #pega o caminho do arquivo
                  path =
"colecao_teste/FSP.{:02d}{:02d}{:02d}.sgml".format(year, month,
day)
                  #lendo o documento
                  file_doc = read_file(path)
                  file_tt = file_doc.read()
                  file_doc.close()
                  all_tags = get_all_tags(file_tt,
except_tegs=['doc','docno'])
                  docs_dataframe = transform_df(file_tt, all_tags)
                  docs_dataframe =
docs_dataframe.rename(columns={'docid':'id'})

                  #conectando e adicionando ao solr
                  list_docs +=  dict_docs(docs_dataframe)
            add_solr(solr_client, list_docs)
```

Apêndice 2: Script para inserir os documentos no Solr, na coleção noticias

Apêndice 3

```
# -*- coding: utf-8 -*-
import pysolr


def define_queries():
      queries = ["text:(+boicote  consumidores~ ! político~)",
                 "text:(terrorista~ +ETA +França~)",
                 "text:(filmes documentários Escócia~ gravação~)",
                 "text:(+resíduos industriais (métodos remoção))",
                 "text:(+desemprego Europa~ números~ taxa~
índice~)",
                 "text:('100° aniversário' centenário '100 anos')",
                 "text:(+espécies +extinção proteger~ Europa~
animal~)",
                 "text:(+greve greve~ causa~ objetivos~ motivo~)",
                 "text:(+ópio produção global~ mundial~ cultivo
papoilas )",
                 "text:(+energia 'crise  energética' energética~
crise combustível~ causa~)"]
```

```
     return queries


def get_data(query, host, collection, port=8983):
     path = 'http://{}:{}/solr/{}'.format(host, port,
collection)
     solr_client = pysolr.Solr(path)
     return solr_client.search(query, rows=100, fl="* score")


def fill_file(queries, host, collection, output_file):
     file_write = open(output_file,"w")
     for index in range(0, len(queries)):
          results = get_data(queries[index],host, collection)
          no_rank = 0
          for result in results:
               file_write.write("{} QO {} {} {}
GustavoM_Leticia\n".format(index+1,
                         result['id'],
                          no_rank,
                         result['score']))
               no_rank += 1
               if no_rank == 100:
                   break

     file_write.close()


queries = define_queries()
fill_file(queries,"localhost","noticias","resultado3.txt")
```

Apêndice 3: Script para consultar os documentos no Solr, na coleção noticias