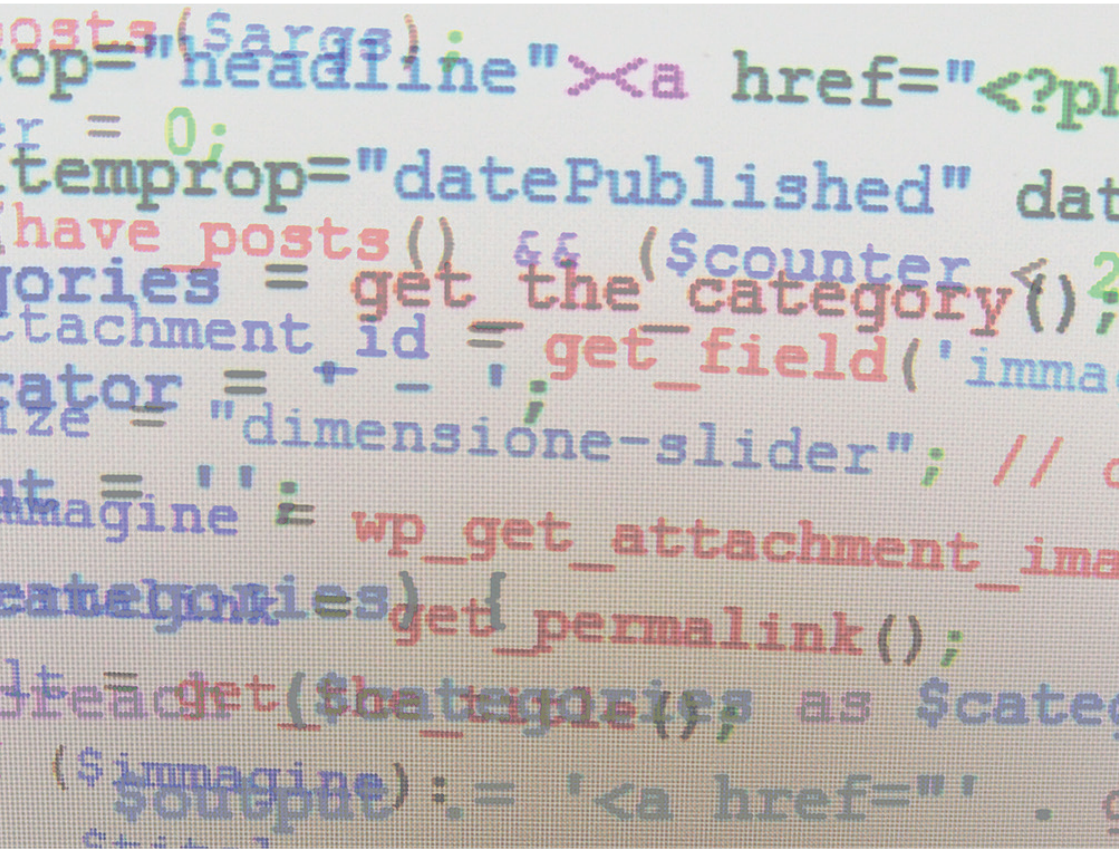
Documentos

ISSN 1677-9274
Dezembro, 2016
144

Plugins do Elasticsearch para tratamento de termos compostos e correção do filtro de sinônimos



Embrapa

ISSN 1677-9274
Dezembro, 2016

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Informática Agropecuária
Ministério da Agricultura, Pecuária e Abastecimento

Documentos 144

Plugins do Elasticsearch para tratamento de termos compostos e correção do filtro de sinônimos

Glauber José Vaz

Embrapa Informática Agropecuária
Campinas, SP
2016

Embrapa Informática Agropecuária
Av. André Tosello, 209 - Barão Geraldo
Caixa Postal 6041 - 13083-886 - Campinas, SP
Fone: (19) 3211-5700
www.embrapa.br/informatica-agropecuaria
SAC: www.embrapa.br/fale-conosco/sac/

1ª edição
publicação digitalizada 2016

Dados Internacionais de Catalogação na Publicação (CIP)
Embrapa Informática Agropecuária

Vaz, Glauber José.

Plugins do Elasticsearch para tratamento de termos compostos e correção do filtro de sinônimos / Glauber José Vaz.-

Campinas : Embrapa Informática Agropecuária, 2016.

28 p. : il. ; cm. - (Documentos / Embrapa Informática Agropecuária, ISSN 1677-9274; 144).

1. Tecnologia da informação. 2. Sistema de recuperação da informação. 3. Sinônimos. 4. Termos compostos. I. Embrapa Informática Agropecuária. II. Título. III. Série.

CDD 004

Autor

Glauber José Vaz
Bacharel e mestre em Ciência da Computação
Analista da Embrapa Informática Agropecuária, Campinas, SP

Apresentação

Sistemas de recuperação de informação são fundamentais para qualquer organização que trabalhe com o uso e a produção de conhecimento de forma intensiva. As demandas por informações de qualidade são, cada vez mais, desafiadoras e importantes para as tomadas de decisão. Neste contexto, os sistemas de recuperação de informação precisam ser constantemente aperfeiçoados.

Na Empresa Brasileira de Pesquisa Agropecuária (Embrapa), diversos sistemas possibilitam a recuperação de informação. Ainfo, Banco de Dados da Pesquisa Agropecuária (BDPA), Sabiia e Alice possibilitam o acesso à relevante produção técnico-científica da agropecuária. Vários outros sistemas e sites da empresa, como Quaesta, portal e intranet, também apresentam mecanismos de busca que facilitam a recuperação de informação. Porém, dada a rápida evolução da ciência da computação e das tecnologias geradas neste campo do conhecimento, é essencial a atualização constante dos profissionais desta área e a incorporação das novas ferramentas e metodologias nas tecnologias da Embrapa.

Elasticsearch é uma destas ferramentas que despontam gerando grande impacto nas organizações e que rapidamente são incorporadas por elas. Utilizada para a construção de mecanismos de busca em textos e em dados estruturados e para análises detalhadas, esta tecnologia vem sendo amplamente empregada para tratar grandes volumes de dados.

É muito comum que soluções computacionais não atendam por completo as demandas específicas dos usuários. Porém, *Elasticsearch* é software livre e os profissionais podem conhecê-lo em profundidade para resolver seus próprios problemas e, quando possível, retornar para a comunidade os avanços alcançados. Este trabalho representa uma contribuição para a comunidade que utiliza Elasticsearch e também para as equipes técnicas responsáveis pelo desenvolvimento dos sistemas de recuperação da informação da Embrapa.

Silvia Maria Fonseca Silveira Massruhá
Chefe-geral
Embrapa Informática Agropecuária

Sumário

Plugins do Elasticsearch para tratamento de termos compostos e correção do filtro de sinônimos

Glauber José Vaz

Introdução

O tratamento de termos compostos e a utilização de sinônimos em sistemas de recuperação de informação possibilitam a obtenção de melhores resultados das buscas. No domínio agropecuário, Marinho et al. (2012) exploraram relações de sinonímias presentes no Thesagro (THESAGRO..., 2006) para um mecanismo de busca envolvendo a produção bibliográfica da Empresa Brasileira de Pesquisa Agropecuária (Embrapa). Estas relações frequentemente envolvem termos compostos por duas ou mais palavras.

Os autores utilizaram Apache Solr (THE APACHE SOFTWARE FOUNDATION, 2016) no desenvolvimento do sistema e exploraram filtros já disponibilizados pela própria tecnologia. Especificamente para o tratamento de termos compostos e de sinônimos, foram usados, respectivamente, os filtros 'ShingleFilter' e 'SynonymFilter'. Zanardo e Vaz (2013) melhoraram o sistema desenvolvendo um componente de software para a tecnologia Solr que faz o tratamento de termos compostos de maneira a utilizar muito menos espaço em memória e também menos tempo de processamento do que o 'ShingleFilter'. Embora o filtro desenvolvido requeira

uma lista de termos considerados relevantes, o que não era necessário no sistema anterior, isso também pode ser vantajoso, pois são considerados termos compostos válidos apenas aqueles fornecidos.

Outra tecnologia mais recente, Elasticsearch, vem crescendo e ocupando espaço no desenvolvimento de aplicações de busca. Assim como Solr, ela é construída em cima do Apache Lucene, considerada a mais avançada biblioteca para a construção de mecanismos de busca em texto. Elasticsearch (GORMLEY; TONG, 2016) é usado para a construção de mecanismos não só de busca em textos, como também de busca estruturada e de análises. É esta combinação que o torna inovador. Além disso, oferece uma interface simples via RESTful API e apresenta uma curva de aprendizado que possibilita que novos usuários possam facilmente começar a utilizá-lo e, em pouco tempo, tornarem-se produtivos.

Neste trabalho, implementamos o mesmo *plugin* para tratamento de termos compostos desenvolvido por Zanardo e Vaz (2013), mas agora para Elasticsearch. Além disso, ainda corrigimos um efeito colateral introduzido pelo filtro de sinônimos quando combinado com o filtro de termos compostos. Este efeito, explicado mais adiante, faz com que respostas relevantes deixem de ser retornadas, afetando assim a qualidade do sistema. Uma alternativa seria desenvolver um novo filtro para tratamento de sinônimos baseado no código do token filter 'Synonym' do Elasticsearch, mas dada a sua complexidade e a facilidade de corrigir o efeito colateral produzido, optamos pelo filtro de correção.

Nas próximas seções os dois *plugins* implementados são explicados em detalhes. Para ilustrarmos os exemplos, utilizamos telas extraídas da aplicação Sense, que oferece um console interativo para submeter diretamente do navegador requisições ao Elasticsearch (GORMLEY; TONG, 2015). Trechos para os quais desejamos chamar atenção são destacados nas diversas figuras, e trechos que não são relevantes para o contexto corrente são removidos. Sense faz parte do Kibana (ELASTIC, 2016), que é uma plataforma de código aberto para visualização e análise de dados armazenados em índices do Elasticsearch.

Plugin CompoundTerms

Para que o *plugin* CompoundTerms seja compreendido, é necessário entender primeiramente como o Elasticsearch funciona. A Figura 1 exibe a configuração de um analisador que pode ser utilizado no processamento de documentos no índice 'teste'. Mais especificamente, exploraremos o analisador 'default_index', pois este caso já é suficiente para explicar tanto o *plugin* CompoundTerms quanto o CorrectSynonyms.

```
PUT /teste
{
  "settings": {
    "number_of_shards" :   1,
    "number_of_replicas" : 1,
    "analysis" : {
      "char_filter": {
        "null_to_empty": {
          "type":       "mapping",
          "mappings": [ "NULL => "]
        },
        "dot_to_blank": {
          "type":       "mapping",
          "mappings": [ ". => \\u0020"]
        }
      },
     "analyzer" : {
        "default_index": {
          "type":           "custom",
          "char_filter":  [ "null_to_empty", "dot_to_blank"],
          "tokenizer":      "uax_url_email",
          "filter":       [ "asciifolding", "lowercase", "trim"]
        }
      }
    }
  }
}
```

Figura 1. Configuração do analisador 'default_index' sem filtros que tratam termos compostos ou correções da aplicação do filtro de sinônimos.

Este analisador é do tipo 'custom' porque o estamos definindo. Os filtros de caracteres, indicados por 'char_filter', analisam cada caractere da entrada para verificar se geram nova saída. O filtro 'null_to_empty' transforma

“NULL” em caracter vazio e o filtro ‘dot_to_blank’ transforma cada ponto final em espaço em branco. O tokenizer é o elemento que quebra a entrada em *tokens*, que correspondem aos termos que serão indexados ou buscados. Os filtros de *token* (*token filters*) processam uma sequência de *tokens* para transformá-la, ou não, em outra sequência. Por exemplo, o ‘asciifolding’ elimina acentuação dos *tokens* e o ‘*lowercase*’ passa todas as letras dos *tokens* para minúsculas. O filtro ‘trim’ elimina espaços em branco nas extremidades dos *tokens*. Os plugins propostos neste trabalho são desse tipo: *token filters*.

Ao realizar no Sense a consulta abaixo, obtém-se a saída mostrada na Figura 2.

```
GET /teste/_analyze?analyzer=default_index&text= recarga do lençol freático
```


```
{
   "tokens": [
      {
         "token": "recarga",
         "start_offset": 1,
         "end_offset": 8,
         "type": "<ALPHANUM>",
         "position": 1
      },
      {
         "token": "do",
         "start_offset": 9,
         "end_offset": 11,
         "type": "<ALPHANUM>",
         "position": 2
      },
      {
         "token": "lencol",
         "start_offset": 12,
         "end_offset": 18,
         "type": "<ALPHANUM>",
         "position": 3
      },
      {
         "token": "freatico",
         "start_offset": 19,
         "end_offset": 27,
         "type": "<ALPHANUM>",
         "position": 4
      }
   ]
}
```


Figura 2. Exemplo de saída obtida com o analisador ‘default_index’ sem filtros que tratam termos compostos ou correções da aplicação do filtro de sinônimos.

A consulta verifica a sequência de *tokens* obtida ao se aplicar o analisador 'default_index' do índice 'teste' à entrada “recarga do lençol freático”. Cada *token* é caracterizado por uma sequência de caracteres que determina o termo (ex: “recarga”), as posições de início e de fim deste termo na entrada (o termo “recarga” corresponde aos caracteres de 1 a 8 na *string* de entrada), o tipo de informação (“<ALPHANUM>” corresponde a um termo alfanumérico) e a posição do termo na *string* de entrada (“recarga” é o

primeiro termo, “do” é o segundo e assim por diante).

No entanto, esta sequência de *tokens* não é capaz de localizar termos compostos, como por exemplo “lençol freático” ou “recarga do lençol freático”. Para que estes termos sejam tratados como *tokens*, é necessário que a saída gerada envolva os elementos presentes na Figura 3.

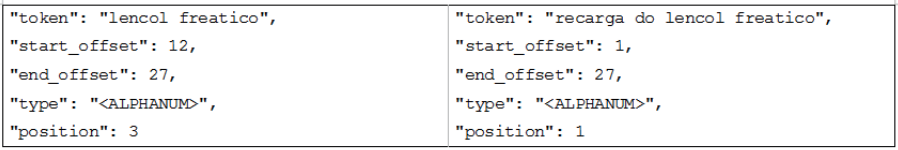


Figura 3. *Tokens* representando termos compostos.

Torna-se necessário então, a inclusão de um novo filtro, o CompoundTerms, conforme especificado na Figura 4. O filtro 'compound' é adicionado à lista de filtros do analisador 'default_index' e seu parâmetro 'shingle' especifica uma lista de arquivos que são utilizados para enumerar os termos considerados relevantes. No exemplo, são considerados dois arquivos: um chamado 'vocabulary.txt' e outro 'synonyms.txt'. O primeiro lista termos de maneira isolada e pode ser usado para, por exemplo, enumerar termos cujas definições foram estabelecidas por um glossário. O segundo apresenta em cada linha termos sinônimos separados por vírgula, conforme mostra a Figura 5. As relações de sinonímia podem ser extraídas de um *thesaurus* como o Thesagro, por exemplo. O uso de letras maiúsculas ou minúsculas não interferem no resultado porque isso é tratado pelos filtros.

Com a utilização deste filtro, a consulta previamente explicada produz a saída mostrada na Figura 6, em que os termos “lencol freatico” e “recarga do lencol freatico” também são tratados como *tokens*. Isso é importante para recuperar elementos indexados por eles. Por exemplo, é possível agora resgatar a definição do termo “recarga do lencol freatico” em um glossário previamente definido ou o sinônimo de “lencol freatico” (“lencol d’agua”). Esses recursos, no entanto, também precisam ser implementados. Neste trabalho não tratamos do glossário porque não exige o desenvolvimento de um novo *plugin*, mas tratamos dos sinônimos.

```
PUT /teste
{
  "settings": {
    :
    "analysis" : {
      :
      "filter": {
        "compound" : {
          "type" : "compound_terms",
          "shingle" : "<PATH>/vocabulary.txt,<PATH>/synonyms.txt"
        }
      },
      "analyzer" : {
        "default_index": {
          "type":           "custom",
          "char_filter":  [ "null_to_empty", "dot_to_blank"],
          "tokenizer":      "uax_url_email",
          "filter":   [ "asciifolding", "lowercase", "trim", "compound"]
        }
      }
    }
  }
}
```

Figura 4. Utilização do filtro CompoundTerms em analisador do Elasticsearch.

:	:
protecao dos recursos hidricos	AGRICULTURA,CIENCIA AGRARIA
recarga do lencol freatico	BAGRE,JUNDIA
rede de canais de irrigacao	LENCOL FREATICO,LENCOL D'AGUA
rendimento de cultura	METANOL,ALCOOL METILICO
residuo de cultura	RADIACAO SOLAR,LUZ SOLAR
:	:
(a)	(b)

Figura 5. Exemplos de termos listados de maneira isolada (a) ou com seus sinônimos (b).

```
{
  "tokens": [
    {
      "token": "recarga",
      "start_offset": 1,
      "end_offset": 8,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "recarga do lencol freatico",
      "start_offset": 1,
      "end_offset": 27,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "do",
      "start_offset": 9,
      "end_offset": 11,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "lencol",
      "start_offset": 12,
      "end_offset": 18,
      "type": "<ALPHANUM>",
      "position": 3
    },
    {
      "token": "lencol freatico",
      "start_offset": 12,
      "end_offset": 27,
      "type": "<ALPHANUM>",
      "position": 3
    },
    {
      "token": "freatico",
      "start_offset": 19,
      "end_offset": 27,
      "type": "<ALPHANUM>",
      "position": 4
    }
  ]
}
```


Figura 6. Saída de um analisador que usa o filtro CompoundTerms.

Plugin CorrectSynonyms

O *plugin* CompoundTerms viabiliza a manipulação de *tokens* formados por várias palavras, desde que relacionados como relevantes. O tratamento de sinônimos de termos compostos, por exemplo, só é possível por conta do CompoundTerms. Porém, a combinação deste com o filtro de sinônimos já disponibilizado pelo Elasticsearch acarreta em um efeito colateral que afeta as posições (atributo 'position') dos *tokens*. O filtro CorrectSynonyms reverte este efeito.

Em primeiro lugar, aplicamos o filtro de sinônimos da própria ferramenta, conforme pode ser observado na Figura 7. Ele é aplicado após o CompoundTerms justamente porque deseja-se tratar sinônimos de termos compostos. Em sua configuração, utiliza o arquivo ilustrado na Figura 5(b) com termos sinônimos separados por vírgula. Também define que não importa se as letras são maiúsculas ou minúsculas no arquivo ("*igno-*

re_case":true). A configuração de *tokenizer* ("*tokenizer*": "*keyword*") determina que cada linha do arquivo lido deve ser processado considerando-se apenas vírgulas como separadores. Por exemplo, a linha "RADIACAO SOLAR,LUZ SOLAR" deve produzir *tokens* "radiacao solar" e "luz solar", mas não "radiacao" ou "luz".

```
PUT /teste
{
  "settings": {
    :
    "analysis" : {
      :
      "filter": {
        "compound" : {
          :
        },
        "synonym_filter":{
          "type": "synonym",
          "synonyms_path":"<PATH>/synonyms.txt",
          "ignore_case":true,
          "tokenizer": "keyword"
        }
      },
      "analyzer" : {
        "default_index": {
          "type":          "custom",
          "char_filter": [ "null_to_empty", "dot_to_blank"],
          "tokenizer":     "uax_url_email",
          "filter":      [ "asciifolding", "lowercase", "trim",
                                          "compound", "synonym_filter"]
        }
      }
    }
  }
}
```

Figura 7. Utilização do filtro de sinônimos em analisador do Elasticsearch.

Com o uso do filtro de sinônimos, a saída gerada para a mesma consulta "recarga do lencol freatico" gera um *token* adicional "lencol d'agua", uma vez que este termo é sinônimo de "lencol freatico". A Figura 8 mostra que o filtro, além de incluir este novo *token*, altera o tipo dos *tokens* sinônimos para "SYNONYM". Porém, também gera um efeito indesejável: as posições

dos *tokens* a partir dos sinônimos são ajustadas para valores incorretos. No exemplo, os *tokens* “lencol freatico”, “lencol d'agua” e “freatico” deveriam estar, respectivamente, nas posições 3, 3 e 4, não em 4, 4 e 5.


```
{
  "tokens": [
    {
      "token": "recarga",
      "start_offset": 1,
      "end_offset": 8,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "recarga do lencol freatico",
      "start_offset": 1,
      "end_offset": 27,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "do",
      "start_offset": 9,
      "end_offset": 11,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "lencol",
      "start_offset": 12,
      "end_offset": 18,
      "type": "<ALPHANUM>",
      "position": 3
    },
    {
      "token": "lencol freatico",
      "start_offset": 12,
      "end_offset": 27,
      "type": "SYNONYM",
      "position": 4
    },
    {
      "token": "lencol d'agua",
      "start_offset": 12,
      "end_offset": 27,
      "type": "SYNONYM",
      "position": 4
    },
    {
      "token": "freatico",
      "start_offset": 19,
      "end_offset": 27,
      "type": "<ALPHANUM>",
      "position": 5
    }
  ]
}
```


Figura 8. Saída de um analisador que usa o filtro de sinônimos do Elasticsearch.

Um problema gerado por uma saída como esta é que as buscas podem fornecer resultados imprecisos, especialmente as que tratam de consultas a frases. Nestes casos, utilizam-se aspas duplas para envolver a *string* de entrada, determinando-se assim que o interesse não está na simples presença dos termos nos documentos, mas também na própria sequência em que aparecem na consulta.

Por exemplo, ao indexar um documento que contenha o trecho “recarga do lençol freático e nascentes” com o analisador 'default_index', o sistema estabeleceria as posições dos termos da seguinte maneira:

```
(“recarga”,1);(“recarga do lencol freatico”,1);(“do”,2);(“lencol”,3);
(“lencol freatico”,4);(“lencol d’agua”,4);(“freatico”,5);(“e”,6);(“na
scentes”,7)
```

Mesmo que se repita na consulta este mesmo trecho delimitado por aspas duplas, o documento mencionado não é recuperado porque os termos não estão nas posições corretas. Ao realizar a busca, o sistema procuraria por uma sequência de *tokens* em posições sequenciais como em:

```
("recarga",1);("do",2);("lencol",3);("freatico",4);("e",5);
("nascentes",6)
```

No entanto, o documento não está indexado desta maneira. O elemento “freatico” aparece na posição 5, duas posições à frente de “lencol”, quando deveria estar a apenas uma. Este erro foi introduzido pelo filtro de sinônimos e nossa solução foi corrigi-lo construindo um novo *plugin*.

Para ilustrar melhor os casos em que este erro ocorre, analisamos a saída, na Figura 9, para a indexação do seguinte trecho:

```
GET /teste/_analyze?analyzer=default_index&text=agricultura alcool
metilico lencol freatico bagre
```

```
{
   "tokens": [
      {
         "token": "agricultura",
         "start_offset": 1,
         "end_offset": 12,
         "type": "SYNONYM",
         "position": 1
      },
      {
         "token": "ciencia agraria",
         "start_offset": 1,
         "end_offset": 12,
         "type": "SYNONYM",
         "position": 1
      },
      {
         "token": "alcool",
         "start_offset": 13,
         "end_offset": 19,
         "type": "<ALPHANUM>",
         "position": 2
      },
      {
         "token": "metanol",
         "start_offset": 13,
         "end_offset": 28,
         "type": "SYNONYM",
         "position": 3
      },
      {
         "token": "alcool metilico",
         "start_offset": 13,
         "end_offset": 28,
         "type": "SYNONYM",
         "position": 3
      },
      {
         "token": "metilico",
         "start_offset": 20,
         "end_offset": 28,
         "type": "<ALPHANUM>",
         "position": 4
      },
      {
         "token": "lencol",
         "start_offset": 29,
         "end_offset": 35,
         "type": "<ALPHANUM>",
         "position": 5
      },
      {
         "token": "lencol freatico",
         "start_offset": 29,
         "end_offset": 44,
         "type": "SYNONYM",
         "position": 6
      },
      {
         "token": "lencol d'agua",
         "start_offset": 29,
         "end_offset": 44,
         "type": "SYNONYM",
         "position": 6
      },
      {
         "token": "freatico",
         "start_offset": 36,
         "end_offset": 44,
         "type": "<ALPHANUM>",
         "position": 7
      },
      {
         "token": "bagre",
         "start_offset": 45,
         "end_offset": 50,
         "type": "SYNONYM",
         "position": 8
      },
      {
         "token": "jundia",
         "start_offset": 45,
         "end_offset": 50,
         "type": "SYNONYM",
         "position": 8
      }
   ]
}
```


Figura 9. Erros de posições introduzidos pelo filtro de sinônimos do Elasticsearch.

Na Figura 9, observamos que o filtro não introduz erros nas posições de *tokens* nas situações em que há relações de sinonímia entre termos simples, como em (“bagre”, “jundia”) ou quando um termo simples do trecho indexado tem sinônimos compostos, como em (“agricultura”, “ciencia agraria”). Porém, quando um termo composto sendo indexado apresenta sinônimos, sejam eles compostos ou simples, surgem erros de posições dos *tokens*, como, por exemplo, nos casos (“alcool metilico”, “alcool”) e (“lençol freatico”, “lencol d'agua”). No primeiro, “metanol” e “alcool metilico” deveriam estar na mesma posição 2 ocupada por “alcool”. No segundo, tanto “lencol freatico” quanto “lencol d'agua” deveriam ocupar a mesma posição de “lencol”. Nestas duas situações, os *tokens* que vêm depois também são afetados, embora as distâncias entre as posições permaneçam corretas.

O filtro CorrectSynonyms faz exatamente o ajuste das posições para os valores corretos. Com isso, consultas a frases, delimitadas por aspas duplas, passam a apresentar resultados mais precisos. A Figura 10 exibe a saída do analisador que inclui o filtro CorrectSynonyms e mostra que a saída desejada é obtida devido ao seu uso. A Figura 11 mostra como ele deve ser configurado no analisador e como deve ser aplicado, logo após o filtro de sinônimos.

```
{
   "tokens": [
      {
         "token": "agricultura",
         "start_offset": 1,
         "end_offset": 12,
         "type": "SYNONYM",
         "position": 1
      },
      {
         "token": "ciencia agraria",
         "start_offset": 1,
         "end_offset": 12,
         "type": "SYNONYM",
         "position": 1
      },
      {
         "token": "alcool",
         "start_offset": 13,
         "end_offset": 19,
         "type": "<ALPHANUM>",
         "position": 2
      },
      {
         "token": "metanol",
         "start_offset": 13,
         "end_offset": 28,
         "type": "SYNONYM",
         "position": 2
      },
      {
         "token": "alcool metilico",
         "start_offset": 13,
         "end_offset": 28,
         "type": "SYNONYM",
         "position": 2
      },
      {
         "token": "metilico",
         "start_offset": 20,
         "end_offset": 28,
         "type": "<ALPHANUM>",
         "position": 3
      },
      {
         "token": "lencol",
         "start_offset": 29,
         "end_offset": 35,
         "type": "<ALPHANUM>",
         "position": 4
      },
      {
         "token": "lencol freatico",
         "start_offset": 29,
         "end_offset": 44,
         "type": "SYNONYM",
         "position": 4
      },
      {
         "token": "lencol d'agua",
         "start_offset": 29,
         "end_offset": 44,
         "type": "SYNONYM",
         "position": 4
      },
      {
         "token": "freatico",
         "start_offset": 36,
         "end_offset": 44,
         "type": "<ALPHANUM>",
         "position": 5
      },
      {
         "token": "bagre",
         "start_offset": 45,
         "end_offset": 50,
         "type": "SYNONYM",
         "position": 6
      },
      {
         "token": "jundia",
         "start_offset": 45,
         "end_offset": 50,
         "type": "SYNONYM",
         "position": 6
      }
   ]
}
```


Figura 10. Erros de posições introduzidos pelo filtro de sinônimos do Elasticsearch.

```
PUT /teste
{
  "settings": {
    :
    "analysis" : {
      :
      "filter": {
        "compound" : {
          :
        },
        "synonym_filter":{
          :
        },
        "correct_syn":{
          "type":"correct_synonyms"
        }
      },
      "analyzer" : {
        "default_index": {
          "type":           "custom",
          "char_filter":  [ "null_to_empty", "dot_to_blank"],
          "tokenizer":      "uax_url_email",
          "filter":       [ "asciifolding", "lowercase", "trim",
                            "compound", "synonym_filter", "correct_syn"]
        }
      }
    }
  }
}
```

Figura 11. Utilização do filtro CorrectSynonyms em analisador do Elasticsearch.

Implementação dos *plugins*

Os plugins foram implementados conforme passos indicados em KAREVOL (2013). A estrutura do *plugin* CompoundTerms é apresentada na Figura 12. O CorrectSynonyms apresenta a mesma estrutura, mas com diferença nos nomes dos arquivos para o diretório 'java', conforme Figura 13. O arquivo 'pom.xml' fornece as configurações para o Maven, que é uma ferramenta de automação de compilação para projetos de software. O arquivo '*plugin*.xml' configura como o *plugin* será empacotado e o arquivo 'es-*plugin*.properties' informa ao Elasticsearch o nome do *plugin* com

as seguintes linhas, respectivamente, para o CompoundTerns e para o CorrectSynonyms:

```
plugin=elasticsearch.compoundterms.CompoundTermsPlugin
plugin=elasticsearch.correctsynonyms.CorrectSynonymsPlugin
```

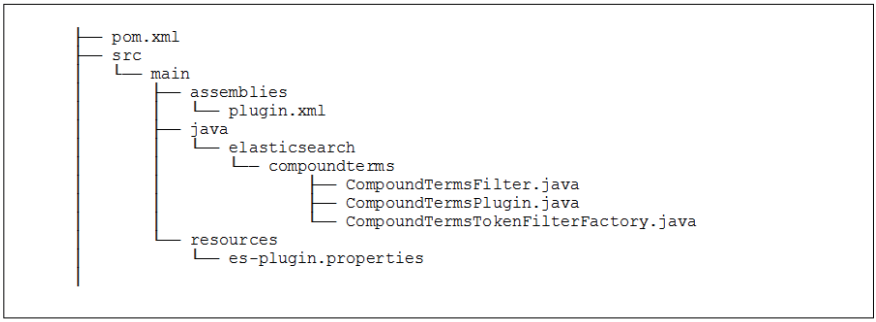


Figura 12. Estrutura do *plugin* CompoundTerms.

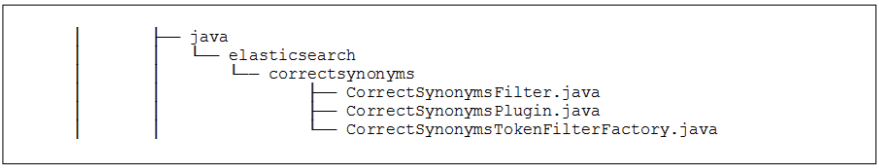


Figura 13. Estrutura do *plugin* CorrectSynonyms.

Os arquivos '**Plugin*.java' definem classes herdeiras de 'org.elasticsearch.plugins.AbstractPlugin', que corresponde à classe base de um *plugin*. A Figura 14 exibe o código de 'CorrectSynonymsPlugin.java', cuja estrutura é análoga ao do 'CompoundTermsPlugin.java'.

Os arquivos '*TokenFilterFactory.java' definem classes herdeiras de 'org.elasticsearch.index.analysis.AbstractTokenFilterFactory' que criam instâncias dos filtros e tratam seus parâmetros. O nome deste arquivo está relacionado ao campo 'type' utilizado na configuração de filtro no Elasticsearch. Conforme Figuras 4 e 11, CompoundTerms é do tipo 'compound_terms' e CorrectSynonyms do tipo 'correct_synonyms'. O primeiro está associado à classe 'CompoundTermsTokenFilterFactory' e o segundo a 'CorrectSynonymsTokenFilterFactory'. Se, por exemplo, alguém declarasse o tipo 'compound-terms', com '-' no lugar de '_', a classe procurada seria 'org.elasticsearch.index.analysis.compound-terms.Compound-

termsTokenFilterFactory', o que provocaria um erro, uma vez que a classe não existe. Portanto, os arquivos '*TokenFilterFactory.java' precisam ter um nome correspondente ao valor utilizado no atributo 'type' do filtro.

```
package elasticsearch.correctsynonyms;
import org.elasticsearch.plugins.AbstractPlugin;
public class CorrectSynonymsPlugin extends AbstractPlugin {
   @Override
   public String name() {
       return "correct-synonyms-plugin";
   }
   @Override
   public String description() {
       return "TokenFilter to correct token position after Synonyms Token Filter";
   }
}
```

Figura 14. Código de 'CorrectSynonymsPlugin.java'.

As Figuras 15 e 16 exibem os principais trechos dos códigos dos arquivos '*TokenFilterFactory.java'. Ambas exibem construtores e implementações para o método *create*(*Tokenstream*) que retorna uma instância da classe '**Filter*'. A diferença é que o filtro CompoundTerms tem o parâmetro 'shingle' como pôde ser observado na Figura 4. Este parâmetro corresponde a uma lista de nomes de arquivos que são processados pelo método *processShingle*(*String*) e populam um atributo 'compounds' do tipo 'Map' com termos compostos que são considerados relevantes. Já o filtro CorrectSynonyms não possui parâmetro, como pôde ser observado na Figura 11.

Os arquivos '**Filter*' definem as classes que determinam os comportamentos dos filtros. Estas classes herdam 'org.apache.lucene.analysis.TokenFilter', base dos filtros que recebem uma sequência de *tokens* na entrada e retorna uma nova sequência resultante do processamento realizado pelo filtro.

Tanto a classe 'CompoundTermsFilter' quanto a 'CorrectSynonymsFilter', filhas de 'TokenFilter', implementam construtor e o método *boolean incrementToken()*. A primeira é bastante complexa e não é escopo deste documento expor seus detalhes, porém vale destacar que é uma adaptação para o Elasticsearch do filtro desenvolvido originalmente para Solr por Zanardo e Vaz (2013). Este, por sua vez, foi baseado no filtro

'ShingleFilter'. Já a classe 'CorrectSynonymsFilter' pode ser rapidamente explicada devido à sua simplicidade. Este, afinal, foi o motivo pela decisão de se desenvolver um novo filtro que corrige os efeitos colaterais do 'SynonymFilter' do Elasticsearch, em vez de se construir um novo filtro para tratar sinônimos que não apresentasse esses efeitos.

```
public class CompoundTermsTokenFilterFactory extends AbstractTokenFilterFactory {

  private Map<String,List<String>> compounds = new TreeMap<String,List<String>>();

  @Inject
  public CompoundTermsTokenFilterFactory(Index index, @IndexSettings Settings
       indexSettings, @Assisted String name, @Assisted Settings settings) {
      super(index, indexSettings, name, settings);
      String shingle = settings.get("shingle");
      processShingle(shingle);
  }

  @Override
  public TokenStream create(TokenStream tokenStream) {
      return new CompoundTermsFilter(Version.LATEST, tokenStream, compounds);
  }

  void processShingle(String nameFile){
      …
  }
}
```

Figura 15. Código de 'CompoundTermsTokenFilterFactory.java'.

```
public class CorrectSynonymsTokenFilterFactory extends AbstractTokenFilterFactory {
  @Inject
  public CorrectSynonymsTokenFilterFactory(Index index, @IndexSettings Settings
              indexSettings, @Assisted String name, @Assisted Settings settings) {
      super(index, indexSettings, name, settings);
  }


  @Override
  public TokenStream create(TokenStream tokenStream) {
      return new CorrectSynonymsFilter(Version.LATEST, tokenStream);
  }
}
```

Figura 16. Código de 'CorrectSynonymsTokenFilterFactory.java'.

A classe 'CorrectSynonymsFilter' verifica, para cada *token* do tipo 'SYNONYM', a posição do seu primeiro caractere na *string* de entrada, representada por 'start_offset'. Se ela for igual à posição do primeiro carac-

tere do *token* anterior, ambos os *tokens* devem estar na mesma posição, representada por 'position'. Se estes valores são diferentes, então o filtro os ajusta de maneira que o *token* analisado recebe um valor de posição igual à do anterior. Este procedimento pode ser verificado observando-se as Figuras 9 e 10.

Os exemplos de saída dos filtros de *tokens* que exibimos anteriormente enumeram os seguintes campos: '*token*', 'start_offset', 'end_offset', 'type' e 'position'. Para este filtro, são relevantes 'start_offset', que determina a posição do primeiro caractere do *token* na *string* de entrada, 'type', que determina o tipo do *token*, e 'position', que determina a posição do *token* na entrada. No entanto, o valor de 'position' não é diretamente armazenado nos *tokens*. O que se armazena é o incremento de posições do *token* corrente em relação ao anterior. Então para que dois termos ocupem uma mesma posição, este incremento deve ser zero.

A Figura 17 exibe o código de 'CorrectSynonymsFilter.java'. O *boolean incrementToken()*, seu principal método, analisa os *tokens* em sequência. Quando ainda há *tokens* para analisar, retorna *true*. Caso contrário, retorna *false*. Este método utiliza os seguintes atributos:

- 'typeAtt': contém o 'type' do *token*;

- 'posIncrAtt': armazena o incremento de posições do *token* corrente em relação ao anterior;

- 'offsetAtt': contém 'start_offset' e 'end_offset' do *token*, mas apenas o primeiro é relevante no filtro CorrectSynonyms;

- 'previousStartOffset': armazena o valor de 'start-offset' do *token* previamente processado.

As variáveis locais 'currentType', 'currentPositionIncrement' e 'currentStartOffset' são usadas para manipular as informações dos três primeiros atributos do *token* corrente.

A instrução condicional da Figura 17 verifica se o ‘start_offset’ do *token* corrente é igual ao do *token* anterior, se o *token* corrente é do tipo ‘SYNONYM’ e se o incremento de posições é igual a 1. Se todas estas condições ocorrem, há erro de posição, pois o incremento de posições deveria ser zero, uma vez que o *token* corrente e o anterior devem ocupar a mesma posição, já que são sinônimos e começam com o mesmo caractere. Neste caso, o incremento de posições do *token* corrente deve

```
public final class CorrectSynonymsFilter extends TokenFilter {
   private final PositionIncrementAttribute posIncrAtt =
                                addAttribute(PositionIncrementAttribute.class);
   private final TypeAttribute typeAtt = addAttribute(TypeAttribute.class);
   private final OffsetAttribute offsetAtt = addAttribute(OffsetAttribute.class);
   private int previousStartOffset = -1;

   public CorrectSynonymsFilter(Version matchVersion, TokenStream input)       {
       super(input);
   }

   @Override
   public boolean incrementToken() throws IOException {
       if  (!input.incrementToken()) {
          return false;
       }
       else {
          String currentType = typeAtt.type();
          int currentPositionIncrement = posIncrAtt.getPositionIncrement();
          int currentStartOffset = offsetAtt.startOffset();

          if ((currentStartOffset == previousStartOffset) &&
               (currentPositionIncrement == 1) && (currentType.equals("SYNONYM"))){
                   posIncrAtt.setPositionIncrement(0);
          }
          previousStartOffset = currentStartOffset;
       }
       return true;
   }
}
```

Figura 17. Código de ‘CorrectSynonymsFilter.java’.

ser ajustado para zero.

Portanto, o filtro CorrectSynonyms apenas atualiza o incremento de posições de *tokens* em relação a *tokens* anteriores nas situações em que houve erro de posicionamento introduzido pelo filtro de sinônimos.

Conclusões

Este trabalho apresenta a importância de dois novos *plugins* para o Elasticsearch e explica sua implementação e a maneira como devem ser usados em um sistema de recuperação de informação. O uso destes dois *plugins*, CompoundTerms e CorrectSynonyms, é essencial em aplicações que demandam o tratamento de termos compostos previamente determinados e também para que o tratamento de sinonímias não comprometa a qualidade das buscas realizadas no sistema.

Referências

ELASTIC. **Kibana user guide [4.6]:** introduction. 2016. Disponível em: <https://www.elastic.co/guide/en/kibana/current/introduction.html>. Acesso em: 26 set. 2106.

GORMLEY, C.; TONG, Z. **Elasticsearch**: the definitive guide [2.x]. 2015. Disponível em: <https://www.elastic.co/guide/en/elasticsearch/guide/current/index.html> Acesso em: 26 set. 2106.

KAREVOL, N. Writing an elasticsearch *plugin*: getting started. 2013. Disponível em: <https://www.elastic.co/blog/found-writing-a-*plugin*>. Acesso em: 26 set. 2016.

MARINHO, I. J. P.; CARDONE, H. T. M.; VAZ, G. J. Evolução do mecanismo de busca do Ainfo-Consulta com uso de thesaurus agropecuário. In: CONGRESSO INTERINSTITUCIONAL DE INICIAÇÃO CIENTÍFICA, 6., 2012, Jaguariúna. **Anais**... Jaguariúna: Embrapa; ITAL, 2012. p. 1-9. CIIC.

THE APACHE SOFTWARE FOUNDATION. **Apache Solr.** Disponível em: <http://lucene.apache.org/solr/>. Acesso em: 26 set. 2016.

THESAGRO - Thesaurus Agrícola Nacional. Brasília, DF, 2006. 278 p.

ZANARDO, D. F.; VAZ, G. J. Tratamento eficiente de termos compostos na tecnologia Solr. In: MOSTRA DE ESTAGIÁRIOS E BOLSISTAS DA EMBRAPA INFORMÁTICA AGROPECUÁRIA, 9., 2013, Campinas. **Resumos**... Brasília, DF: Embrapa, 2013. p. 25-27.

Embrapa

Informática Agropecuária

MINISTÉRIO DA
AGRICULTURA, PECUÁRIA
E ABASTECIMENTO

BRASIL
GOVERNO FEDERAL

CGPE 13449