Análise de dados da rede social Twitter

Practical work on collecting data from twitter, storing data, text-processing and some term frequencies analysis.

Análise de dados da rede social Twitter - Um trabalho prático de introdução para a disciplina banco de dados NoSQL.

Autor: Francisco M. Moura

Github: Social Media data Analysis - Análise de dados de média social

Gitbook: Análise de dados da rede social Twitter

---

1. Análise de dados da rede social Twitter

1.1. Introdução

Este trabalho é parte da disciplina **Banco de dados não relacionais**, cursada no Instituto de Educação Continuada (IEC) da PUC Minas, no curso de pós-graduação Ciência de dados e big data, no segundo semestre de 2016.

O professor Grabriel Campos [GCOUTI] apresentou este desafio como introdução da disciplina.

O trabalho prático foi desenvolvido com dados coletados na rede social Twitter, através das APIs de Streaming [STREAMING-API].

1.1. Contexto e aplicações

Saúde pública é uma área que demanda bastante atenção das instituições públicas ligadas ao Ministério da Saúde e autoridades políticas.
Fatos como a discussão no senado federal brasileiro da ementa que pretende regular a interrupção voluntária da gravidez, dentro das doze primeiras semanas de gestação, pelo sistema único de saúde; nova ameaça do vírus zika; crise econômica aumentando os casos de ansiedade e depressão e outros de interesse da temática saúde pública são assuntos comentados frequentemente pelos usuários da rede social Twitter.

Com esta motivação, pretende-se inicialmente coletar um volume de tweets que permita gerar uma massa de dados para análise que possa ser empregada em uma arquitetura de big data para tal objetivo.
Assim, definiu-se que os conjuntos de dados coletados sejam armazenados no banco de dados NoSQL MongoDB.

2. Metodologia e desenvolvimento do trabalho prático

O trabalho prático foi desenvolvido com dados coletados na rede social do Twitter, utilizando a API de Streaming, conhecidos por tweets [KUMAR2014TWITTE] e [BONZANINI2016MASTERING].

Foram utilizadas as linguagens de programação Python e JavaScript e o banco de dados NoSQL MongoDB como componentes constituíntes da arquitetura do ambiente de análise de dados em grande quantidades.

2.1. Coleta dos dados

Foram coletados um pouco mais de 1 milhão de tweets (1.083.884), entre os dias 30 de novembro/2016 e 21 de dezembro/2016, dando ênfase na temática "**Saúde pública**".

Os termos utilizados para a coleta dos tweets, chamados de *tracker words* pela API de Streaming do Twitter, foram:

```
tracker_words = ['estupro', 'denúncia', 'vítima', 'abuso sexual',
                 'assédio sexual', 'violência sexual', 'dengue', 'gripe',
                 'resfriado', 'malária', 'febre', 'chikungunya', 'zika',
                 'vírus', 'mosquito', 'Aedes aegypti', 'depressão',
                 'ansiedade']
```

`Arquivo de código-fonte: src/crawler_tweets_saude_publica.py`

O código-fonte completo utilizado para a coleta dos tweets encontra-se disponível no github em crawler_tweets_saude_publica.py.
Foi escolhida a estratégia de salvar os dados em arquivos no formato JSON (JavaScript Object Notation - Notação de objetos JavaScript) no sistema de arquivos do sistema operacional, pelo fato dos dados recebidos já se encontrarem nesse formato e por permitir o acompanhamento da evolução da coleta utilizando comandos do bash em sistema Unix like, bem como monitorar a execução dos scripts python devido a limitação de conectividade de rede ou queda do serviço disponibilizado pelo Twitter, facilidade de

importação para o banco de dados NoSQL MongoDB, armazenamento e reutilização (data lake) para trabalhos futuros.

Com o comando **wc** (contador de palavras, linhas, caracteres e bytes) é possível acompanhar a evolução da coleta, por exemplo inspecionando o número de linhas contidas no arquivo.

> ```
> wc -l *.json
> ```
>
> Saída:
>
> ```
> 1111 tweets.saude.coletiva-01.json
> 2424 tweets.saude.coletiva-02.json
> ...
> 93801 tweets.saude.coletiva-22.json
> 1083884 total
> ```

2.2. Ingestão dos dados no MongoDB

A ingestão dos dados no banco MongoDB é realizada com extrema facilidade, visto que os dados contidos nos arquivos JSON estão no formato que o MongoDB utiliza em sua estrutura de armazenamento.

Com o utilitário **mongoimport** os dados podem ser carregados para o processamento no MongoDB.

> Comando e sintaxe:
>
> ```
> mongoimport -d <db_name> -c <collection_name> --file <path/file_na
> me>
>
> Onde:
>    -d: nome da base de dados (esquema no modelo relacional)
>    -c: nome da coleção (tabela no modelo relacional)
> ```

Para cada arquivo, o comando será:

> ```
> mongoimport -d tweets_raw -c saude_coletiva --file tweets.saude.co
> letiva-xx.json
> ```

De forma a automatizar a importação utilizou-se um script shell (***src/mongoimport.sh***), como segue:

```
for file_name in $(ls -d tweets.saude.coletiva-*)
```

```
do
    mongoimport -d tweets_raw -c saude_coletiva --file $file_name
    mv $file_name "proc-$file_name"
done
```

`Arquivo de código-fonte: src/mongoimport.sh`

O funcionamento do script é tal que após a ingestão dos dos no MongoDB, o arquivo é renomeado para indicar que foi processado e futuras re-execuções não duplique os dados na coleção armazenada no banco.

Para averiguar o sucesso da ingestão dos tweets no MongoDB, utilizou-se os seguintes comandos:

> Shell:
>
> ```
> wc -l proc*
> ```
>
> MongoDB:
>
> ```
> db.saude_coletiva.count()
> ```

O resultado da execução destes dois comando devem ser idênticas, ou seja, o mesmo valor numérico.

2.3. Otimização das consultas: Organização da coleção

Visando otimizar o tempo de realização das consultas no MongoDB, convertemos o valor String do atributo "*created_at*" em ***ISODate***, o adicionamos em um novo atributo "*time_stamp*" e atualizamos a coleção.

Segundo [KUMAR2014TWITTE] o atributo "*created_at*" e uma informação de data em formato legível para o ser humano. No entanto, realizar conversão a cada consulta adiciona um custo computacional desnecessário.

Com o código a seguir, realizou-se a operação de conversão e adição do novo atributo "*time_stamp*" do tipo ***ISODate*** (***src/otimizar-colecao.js***) à coleção:

```
db.saude_coletiva.find().forEach(function(doc){
    //save the time string in ISODate.
    doc.timestamp = new Date(Date.parse(doc.created_at));

    //save our modifications
    db.saude_coletiva.save(doc);
});
```

`Arquivo de código-fonte: src/otimizar-colecao.js`

Este arquivo pode ser invocado e executado diretamente no shell MongoDB com o seguinto comando:

> ```
> load("/path/src/02-otimizar-colecao.js")
> ```

Consulte o apêndice para mais detalhes.

Com esta operação realizou-se a primeira etapa necessária para a otimização das consultas.

Vejamos os resultados comparativos do tempo de execução de uma das principais consultas que serão submetidas ao MongoDB para extração de resultados. A consulta é do tipo agregação (db.collection.aggregate()).

2.3.1. Consulta volume por dia no atributo time_stamp do tipo ISODate

```
db.saude_coletiva.aggregate([
    {
        $group: {
            _id: {
                time_stamp: {
                    ano: {$year: "$time_stamp"},
                    mes: {$month: "$time_stamp"},
                    dia: {$dayOfMonth: "$time_stamp"}
                }
            },
            qtde: {$sum: 1}
        }
    },
    {$sort: {_id: 1}}
])
```

`Arquivo de código-fonte: src/volume-tweets-dia-timestamp.js`

A consulta acima (volume-tweets-dia-timestamp.js) utiliza o atributo *time_stamp* criado especialmente para otimizar o tempo de execução das operações computacionais no MongoDB. Veja abaixo o coparativo entre a consulta que utiliza o atributo previamente preparado (time_stamp) com a consulta que utiliza o atributo do tipo String (created_at).

Para obter a informação do tempo de execução de uma consulta do tipo ***db.collection.aggregate*** é necessário procurar no log de execução do *deamon mongod* por `protocol:op_command XXms` .

Log do deamon mongod:

```
command tweets_raw.saude_coletiva appName: "MongoDB Shell" command
:
    aggregate { aggregate: "saude_coletiva",
    pipeline: [ { $group: { _id: { time_stamp: { ano: { $year: "$t
ime_stamp" },
    mes: { $month: "$time_stamp" }, dia: { $dayOfMonth: "$time_sta
mp" } } },
    total: { $sum: 1.0 } } }, { $sort: { _id: 1.0 } } ], cursor: {
} }
    planSummary: COLLSCAN keysExamined:0 docsExamined:1083884 hasS
ortStage:1 cursorExhausted:1
    numYields:8566 nreturned:23 reslen:1882
    locks:{ Global: { acquireCount: { r: 17160 } }, Database: { ac
quireCount: { r: 8580 } },
    Collection: { acquireCount: { r: 8579 } } } protocol:op_comman
d 3809ms
```

2.3.2. Consulta volume por dia no atributo created_at do tipo String

```
db.saude_coletiva.aggregate([
    {
        $match: {
            $or: [
                {"created_at" : {$in : [/Nov/]}},
                {"created_at" : {$in : [/Dec/]}}
            ]
        }
    },
    {
        $group: {
            _id: {
                ano : { $substr : ["$created_at", 26, 4 ] },
                mes : { $substr : ["$created_at", 4, 3 ] },
                dia : { $substr : ["$created_at", 8, 2 ] }
            },
            total: { $sum: 1 }
        }
    },
    {
        $sort: {_id: 1}
```

```
    }
])
```

`Arquivo de código-fonte: src/volume-tweets-dia-created_at.js`

Esta consulta (volume-tweets-dia-created_at.js) foi utilizada somente com o objetivo de gerar um comparativo de tempo de execução entre a consulta anterior.

Como explanado anteriormente, é preciso consultar o log do daemon da engine do MongoDB para obter o tempo de execução, olhando o atributo ``protocol:op_command.

Log do deamon mongod:

```
command tweets_raw.saude_coletiva appName: "MongoDB Shell" command
:
    aggregate { aggregate: "saude_coletiva",
    pipeline: [ { $match: { $or: [ { created_at: { $in: [ /Nov/ ]
} },
    { created_at: { $in: [ /Dec/ ] } } ] } },
    { $group: { _id: { ano: { $substr: [ "$created_at", 26.0, 4.0
] },
    mes: { $substr: [ "$created_at", 4.0, 3.0 ] }, dia: { $substr:
 [ "$created_at", 8.0, 2.0 ] } },
    total: { $sum: 1.0 } } }, { $sort: { _id: 1.0 } } ], cursor: {
} }
    planSummary: COLLSCAN keysExamined:0 docsExamined:1083884 hasS
ortStage:1 cursorExhausted:1
    numYields:8659 nreturned:22 reslen:1694 locks:{ Global: { acqu
ireCount: { r: 17352 } },
    Database: { acquireCount: { r: 8676 } }, Collection: { acquire
Count: { r: 8675 } } }
    protocol:op_command 7183ms
```

2.3.3. Comparativo do tempo de execução entre objetos do tipo ISODate e String

Consulta - sem utilização de indice	Ordenação	Tempo de execução
Atributo time_stamp, tipo ISODate	Ascendente	3.809 ms
Atributo created_at, tipo String	Ascendente	7.183 ms
Atributo time_stamp, tipo ISODate	Descendente	5.412 ms
Atributo created_at, tipo String	Descendente	5.571 ms

Como esperado, as consultas que utilizam o atributo auxiliar time_stamp apresentou melhor desempenho no comparativo com o atributo created_at.

2.4. Otimização das consultas: Criação de índices

Um segundo procedimento necessário para completar a otimização das consultas para o cenário da análise dos dados é a criação de um índice para o atributo time_stamp e outro índice para o atributo created_at.

A criação de índices se faz necessária para evitar a execução padrão das consultas, que comumente são mais demoradas porque executam leitura sobre todos os dados da coleção. Os índices, se bem utilizados, tornam o tempo de execução das consultas melhores porque não são lidos todos os dados da coleção, ou permite que o MongoDB utilize melhores estratégias para execução da consulta submetida.
Com o comando `db.collection.createIndex()` foi criado o índice do tipo Ascendente (ordenando as datas na ordem da menor para a maior) para o atributo time_stamp:

```
> db.saude_coletiva.createIndex({time_stamp: 1})
```

E para o atributo created_at:

```
> db.saude_coletiva.createIndex({created_at: "text"})
```

`Arquivo de código-fonte: src/criar-indices.js`

A criação do índice para o atributo created_at foi diferente porque o MongoDB permite criar o índice do tipo `text` e, assim, realizar buscas dentro da string indexada (src/criar-indices.js).

Vejamos os resultados comparativos da execução da mesmas consultas de agregação utilizadas na seção anterior:

Consulta - com utilização de índice	Ordenação	Tempo de execução
Atributo time_stamp, tipo ISODate	Descendente	4.040 ms
Atributo created_at, tipo String	Descendente	3.699 ms
Atributo time_stamp, tipo ISODate	Ascendente	3.931 ms
Atributo created_at, tipo String	Ascendente	3.670 ms

Como esperado, com a utilização de índices o desempenho do tempo de realização das consultas é

melhorado.

2.5. Pré-processamento de texto

Para calcular a frequência das palavras contidas nos tweets, optou-se pelo uso da linguagem Python para realizar o pré-processamento do texto. Esta etapa consiste na remoção de caracteres de pontuação, artigos, caracteres de controle de fim de linha, etc, produzindo os termos, que são as palavras para análise.
Python, através da biblioteca NLTK e normalizr, oferece bom suporte para esta atividade, produzir os termos, que consiste em separar cada palavras do texto em palavras independentes.
Ainda, a biblioteca NLTK oferece suporte para processar adequadamente os temos como hastag, usuários mencionados nos textos dos tweets, ou seja, estes termos não são descartados e são incluídos no processo de análise.

Os termos produzidos foram armazenados em uma nova coleção no MongoDB, de forma que pode-se aplicar quaisquer funções (Map/Reduce ou Aggregate) para avaliar a ocorrência das palavras no conjunto de dados coletados.

Ao final desta etapa, foram gerados 10.111.657 termos, que foram armazenados no MongoDB, em uma nova coleção.

O código abaixo mostra como realizou-se o pré-processamento (src/termos_frequencia.py):

```
# -*- coding: utf-8 -*-
import string
from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer
import pymongo

# from https://github.com/davidmogar/normalizr
def remove_acentos(tokens=list()):
    """
    Remove acentos dos tokens
    Return: lista de tokens
    """
    import unicodedata
    from normalizr import Normalizr
    normalizr = Normalizr(language='pt')
    normalizations = [ 'remove_accent_marks' ]
    l = list()
    for token in tokens:
```

```
        l.append(normalizr.normalize(token, normalizations))
    return l

# extract from book Mastering social media mining with Python
def process(text, tokenizer=TweetTokenizer(), stopwords=[]):
    """Process the text of a tweet:
    - Lowercase
    - Tokenize
    - Stopword removal
    - Digits removal
    Return: list of strings
    """
    text = text.replace('\n', ' ').replace('\t','').lower()
    tokens = tokenizer.tokenize(text)
    return [tok for tok in tokens if tok not in stopwords and not tok.isdig
it()]

if __name__ == '__main__':
    tweet_tokenizer = TweetTokenizer()
    punctuation = list(string.punctuation)
    stopword_pt = stopwords.words('portuguese')
    stopword_es = stopwords.words('spanish')
    stopword_en = stopwords.words('english')
    stopword_all = stopword_pt + stopword_es + stopword_en
    stopword_all += punctuation + ['rt', 'via', '...', '…']
    # print(stopword_all)
    client = pymongo.MongoClient('localhost', 27017)
    db = client.tweets_raw
    # tokenizar todos os texts dos tweets
    tweets = db.saude_coletiva.find({}, {"_id": 0, "text": 1})
    number_of_tweets = 0
    for text in tweets:
        number_of_tweets += 1
        print('number of tweets processed: {}'.format(number_of_tweets))
        if (text.get('text') != None):
            tokens = process(text.get('text'), tweet_tokenizer, stopword_al
l)
            # print(tokens)
            for token in remove_acentos(tokens):
                db.tweet_terms.insert_one({"termo": token})
```

```
    print("Completed....!!!")
```

`Arquivo de código-fonte: src/termos_frequencia.py`

3. Análises e resultados

Foram extraídas informações do tipo quantidade de tweets coletados por dia (**volume por dia**), quantidade de tweets coletados por hora do dia (**volume por hora do dia**) e os **termos mais frequentes**.

Abaixo são apresentadas as consultas utilizadas para produzir os resultados.

3.1. Volume de informações por dia

Para extrair as informações de volume por dia foi utilizada a seguinte consulta, no atributo *time_stamp* criado para auxiliar no processo de extração de informações (src/volume-tweets-dia-timestamp.js):

```
db.saude_coletiva.aggregate([
    {
        $group: {
            _id: {
                time_stamp: {
                    ano: {$year: "$time_stamp"},
                    mes: {$month: "$time_stamp"},
                    dia: {$dayOfMonth: "$time_stamp"}
                }
            },
            qtde: {$sum: 1}
        }
    },
    {$sort: {_id: 1}}
])
```

`Arquivo de código-fonte: src/otimizar-colecao.js`

Esta consulta se apresentou melhor porque propiciou:

- Ordenar os resultados, recuperando os meses e dias dos meses na ordem natural que utilizamos;
- Durante o processo de conversão do atributo *created_at* de String para ISODate, o novo valor foi

convertido para timestamp do Unix e os valores ausentes (nulos) foram automaticamente convertidos para a data 01/01/1970. Desta forma, podemos identificar os valores ausentes (nulos) e ainda utilizar os índices criados para o atributo.

O seguinte resultado foi observado:

```
{ "ano" : 1970, "mes" :  1, "dia" :  1, "qtde" :   127 }
{ "ano" : 2016, "mes" : 11, "dia" : 30, "qtde" :  5539 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "qtde" : 68913 }
{ "ano" : 2016, "mes" : 12, "dia" :  2, "qtde" : 64647 }
{ "ano" : 2016, "mes" : 12, "dia" :  3, "qtde" : 11940 }
{ "ano" : 2016, "mes" : 12, "dia" :  4, "qtde" : 56176 }
{ "ano" : 2016, "mes" : 12, "dia" :  5, "qtde" : 66367 }
{ "ano" : 2016, "mes" : 12, "dia" :  6, "qtde" : 74058 }
{ "ano" : 2016, "mes" : 12, "dia" :  7, "qtde" : 77099 }
{ "ano" : 2016, "mes" : 12, "dia" :  8, "qtde" : 42883 }
{ "ano" : 2016, "mes" : 12, "dia" :  9, "qtde" : 80286 }
{ "ano" : 2016, "mes" : 12, "dia" : 10, "qtde" : 27062 }
{ "ano" : 2016, "mes" : 12, "dia" : 11, "qtde" :  3114 }
{ "ano" : 2016, "mes" : 12, "dia" : 12, "qtde" : 47414 }
{ "ano" : 2016, "mes" : 12, "dia" : 13, "qtde" : 48933 }
{ "ano" : 2016, "mes" : 12, "dia" : 14, "qtde" : 42136 }
{ "ano" : 2016, "mes" : 12, "dia" : 15, "qtde" : 54430 }
{ "ano" : 2016, "mes" : 12, "dia" : 16, "qtde" : 63114 }
{ "ano" : 2016, "mes" : 12, "dia" : 17, "qtde" : 58169 }
{ "ano" : 2016, "mes" : 12, "dia" : 18, "qtde" : 55070 }
{ "ano" : 2016, "mes" : 12, "dia" : 19, "qtde" : 50387 }
{ "ano" : 2016, "mes" : 12, "dia" : 20, "qtde" : 61496 }
{ "ano" : 2016, "mes" : 12, "dia" : 21, "qtde" : 24524 }
```

3.2. Volume de informações por hora do dia

A seguinte consulta foi utilizada (src/volume-tweets-hora-dia-timestamp.js):

```
db.saude_coletiva.aggregate([
    {
        $group: {
            _id: {
                time_stamp: {
                    ano: {$year: "$time_stamp"},
```

```
                    mes: {$month: "$time_stamp"},
                    dia: {$dayOfMonth: "$time_stamp"},
                    hora: {$hour: "$time_stamp"}
                }
            },
            qtde: {$sum: 1}
        }
    },
    {$sort: {_id: 1}}
])
```

Analogamente à consulta de extração de informações do volume por dia, foi utilizada a mesma estratégia pelos mesmos benefícios e justificativas.

Os 10 primeiros resultados ordenados obtidos são mostrados abaixo. A ordenação é ascendente:

```
{ "ano" : 1970, "mes" :  1, "dia" :  1, "hora" :  0, "qtde" :  127
 }
{ "ano" : 2016, "mes" : 11, "dia" : 30, "hora" : 19, "qtde" :  143
 }
{ "ano" : 2016, "mes" : 11, "dia" : 30, "hora" : 20, "qtde" : 1305
 }
{ "ano" : 2016, "mes" : 11, "dia" : 30, "hora" : 21, "qtde" : 2087
 }
{ "ano" : 2016, "mes" : 11, "dia" : 30, "hora" : 22, "qtde" :  653
 }
{ "ano" : 2016, "mes" : 11, "dia" : 30, "hora" : 23, "qtde" : 1351
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  0, "qtde" : 3365
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  1, "qtde" : 3269
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  2, "qtde" : 2701
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  3, "qtde" : 1975
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  4, "qtde" : 1195
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  5, "qtde" :  829
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  6, "qtde" :  674
```

```
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  7, "qtde" :  860
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  8, "qtde" : 1037
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" :  9, "qtde" : 1113
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" : 10, "qtde" : 1466
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" : 11, "qtde" : 1773
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" : 12, "qtde" : 2445
 }
{ "ano" : 2016, "mes" : 12, "dia" :  1, "hora" : 13, "qtde" : 2692
 }
```

3.3. Termos mais frequentes

Esta análise utilizou o conjunto de dados (termos) produzidos na fase de pré-processamento do texto de cada tweet. Delegou-se a realização desta atividade para o MongoDB porque as bibliotecas da linguagem Python, embora possuam poderosas implementações para esta tarefa, não processaria rapidamente a o grande volume de termos que o conjunto de dados possue.

```
db.tweet_terms.aggregate([
        { $group: { _id: "$termo", qtde: {$sum: 1 } } },
        { $sort: {qtde: -1}}
    ],
    {
        allowDiskUse:true,
        cursor:{}
    }
)
```

Os 10 termos mais frequentes encontrados em todo o conjunto de são mostrados abaixo:

```
{ "ansiedade", "qtde" : 300222 }
{ "e",         "qtde" : 138336 }
{ "depressao", "qtde" : 130645 }
{ "panico",    "qtde" : 122890 }
```

```
{ "mosquito",  "qtde" : 119245 }
{ "sete",      "qtde" : 114520 }
{ "lagoas",    "qtde" : 114277 }
{ "toc",       "qtde" :  98747 }
{ "gripe",     "qtde" :  92321 }
{ "zika",      "qtde" :  90860 }
{ "terapia",   "qtde" :  88550 }
{ "febre",     "qtde" :  56779 }
{ "psicologo", "qtde" :  55597 }
{ "denuncia",  "qtde" :  51634 }
{ "q",         "qtde" :  51426 }
{ "pra",       "qtde" :  51076 }
{ "casais",    "qtde" :  47759 }
{ "ta",        "qtde" :  46695 }
{ "estupro",   "qtde" :  45230 }
{ "sexual",    "qtde" :  43022 }
```

4. Conclusões e trabalhos futuros

Não restam dúvidas de que as redes sociais fornecem uma forma nova de produzir e consumir informações. Redes sociais, como o Twitter, disponibilizam gratuitamente um certo volume dos dados públicos que são gerados e compartilhados a cada instante pelos usuários participantes da rede.
Embora haja uma avalanche de informações sendo geradas, certos desafios precisam ser superados para uma efetiva coleta e qualidade dos dados.
O primeiro desafio é a coleta dos dados pretendidos. Devido às restrições da política de uso da organização gestora da rede social, a forma como estes dados podem ser coletados, bem como os desafios envolvendo a rede de comunicação de dados (internet).
O segundo desafio é conhecer a estrutura, o tipo do dado que é disponibilizado para coleta.
E um terceiro desafio é determinar o que será coletado, embora dada a variedade do que se produz, pode-se demorar obter a quantidade de dados desejada para o trabalho de análise.
Para superar parte dos desafios é necessário mergulhar na leitura e estudos das APIs (Applications Programming Interfaces - Interface de Programação de Aplicativos) que as redes sociais disponibilizam, bem como conhecer trabalhos prévios públicados pela comunicade. Contar com o apoio da comunidade é de fundamental relevância para o bom desenvolvimento do trabalho prático, visto que há várias ferramentas e bibliotecas sendo desenvolvidas para endereçar os desafios que surgem.

Neste trabalho, pode averiguar

Trabalhos futuros:

- Apresentar os resultados obtidos em gráficos e outras ferramentas modernas de visualização de dados, como D3.js;
- Realizar análises avançadas como classificação de texto, por exemplo;
- Observar evolução de ocorrências por regiões, como por exemplo, termos das doenças tropicais (zika, mosquito)

5. Referências

1. **GCOUTI**. Campos, Gabriel. **Github do professor Gabriel Campos**. URL: https://github.com/gcouti. Acessado em: 20/12/2016.
2. **STREAMING-API**. Twitter. **Streaming APIs**. URL: https://dev.twitter.com/streaming/overview. Acessado em: 20/12/2016.
3. **KUMAR2014TWITTER**. Kumar, Shamanth, Fred Morstatter, and Huan Liu. **Twitter data analytics**. New York: Springer, 2014.
4. **BONZANINI2016MASTERING**. Bonzanini, Marco. *Mastering social media mining with Python*. (2016).
5. **SADALAGE2012NOSQL**. SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL distilled: a brief guide to the emerging world of polyglot persistence**. Pearson Education, 2012.
6. **MONGODBV3.4**. MongoDB versão. **Manual do MongoDB versão 3.4**. URL: https://docs.mongodb.com/manual/. Acessado em 20/12/2016.

Apêndice

1. Projeto no github

social-media-analysis

2. Script *src/mongoimport.sh*

Tornar o script executável. No terminal bash do sistema operacional execute o comando:

```
chmod +x 01-mongoimport.sh
```

No terminal bash do sistema operacional execute o script sob o diretório src/ com o seguinte comando:

```
sh 01-mongoimport.sh
```

3. Script src/otimizar-colecao.js

No shell do MongoDB execute o seguinte comando:

```
> load("/complete-path-to/src/02-otimizar-colecao.js")
```