201708 common c...

FINISHED

WAT files - understanding the JSON structure

What this notebook does:

Develops some functions to parse the json records in WAT files, investigating the trees of json keys and locating the URLs of links.

Took 0 sec. Last updated by anonymous at October 14 2017, 10:44:21 AM.

READY

```
%pyspark

import boto
from boto.s3.key import Key
from gzipstream import GzipStreamFile
from pyspark.sql.types import *
import warc

import json

watlist = sc.textFile("s3://commoncrawl/crawl-data/CC-MAIN-2017-04/wat.paths.gz")
watlist.cache()

conn = boto.connect_s3(anon=True, host='s3.amazonaws.com')
bucket = conn.get_bucket('commoncrawl')

def unpack(uri):
    key_ = Key(bucket, uri)
    file_ = warc.WARCFile(fileobj=GzipStreamFile(key_))
    return file_

def mapper(id_, iterator):

    for uri in iterator:
        file = unpack(uri)
        for record in file:
            try:
                yield record['Content-Type']
            except KeyError:
                yield None

nfiles = 16
files = sc.parallelize(watlist.take(nfiles))

ct = files.mapPartitionsWithIndex(mapper)
ct.cache()
print(ct.count())
print(ct.countByValue())
ct.unpersist()
```

```
2621630
defaultdict(<type 'int'>, {'application/warc-fields': 16, 'application/json': 2621614})
PythonRDD[131] at RDD at PythonRDD.scala:48
```

Let's examine a sample of json records: READY

```
%pyspark

from pprint import pprint

def json_mapper(id_, iterator):
    conn = boto.connect_s3(anon=True, host='s3.amazonaws.com')
    bucket = conn.get_bucket('commoncrawl')

    for uri in iterator:
        key_ = Key(bucket, uri)
        file_ = warc.WARCFile(fileobj=GzipStreamFile(key_))

        for record in file_:
            if record['Content-Type'] == 'application/json':
                record = json.loads(record.payload.read())

                try:
                    yield record
                except KeyError:
                    yield None

nrecords = 100
sample = files.\
        mapPartitionsWithSplit(json_mapper).\
        take(nrecords)

pprint(sample[1])
```

READY

```
{u'Container': {u'Compressed': True,
                u'Filename': u'CC-MAIN-20170116095119-00000-ip-10-171-10-70.ec2.internal.wa
rc.gz',
                u'Gzip-Metadata': {u'Deflate-Length': u'393',
                                   u'Footer-Length': u'8',
                                   u'Header-Length': u'10',
                                   u'Inflated-CRC': u'2084246495',
                                   u'Inflated-Length': u'566'},
                u'Offset': u'431'},
 u'Envelope': {u'Actual-Content-Length': u'213',
               u'Block-Digest': u'sha1:T2IJZ2UKM44CKVZAPHICBH7EKI22CVBJ',
               u'Format': u'WARC',
               u'Payload-Metadata': {u'Actual-Content-Type': u'application/http; msgtype=re
quest',
                                     u'HTTP-Request-Metadata': {u'Entity-Digest': u'sha1:3I
42H3S6NNFQ2MSVX7XZKYAYSCX5QBYJ',
                                                                u'Entity-Length': u'0',
                                                                u'Entity-Trailing-Slop-Byte
```

We see that the field `['Envelope']['WARC-Header-Metadata']['WARC-Target-URI']` contains the URI of the current web page: READY

```
%pyspark
```

READY

```
for rec in sample:
    try:
        print(rec['Envelope']['WARC-Header-Metadata']['WARC-Target-URI'])
    except KeyError:
        print("")
```

```
http://03online.com/news/3383
http://03online.com/news/3383
http://03online.com/news/3383
http://03online.com/news/pochemu_sineyut_guby/2013-2-28-4347
http://03online.com/news/pochemu_sineyut_guby/2013-2-28-4347
http://03online.com/news/pochemu_sineyut_guby/2013-2-28-4347
http://03online.com/news/temnye_krugi_pod_glazami/2014-7-17-28934
http://03online.com/news/temnye_krugi_pod_glazami/2014-7-17-28934
http://03online.com/news/temnye_krugi_pod_glazami/2014-7-17-28934
http://05sese.com/news/class/160566.html
http://05sese.com/news/class/160566.html
http://05sese.com/news/class/160566.html
http://08.od.ua/gazovoe_i_otopitelnoe_oborudovanie/kotly_elektricheskie/teplo_holod
http://08.od.ua/gazovoe_i_otopitelnoe_oborudovanie/kotly_elektricheskie/teplo_holod
http://08.od.ua/gazovoe_i_otopitelnoe_oborudovanie/kotly_elektricheskie/teplo_holod
http://08.od.ua/turisticheskie_uslugi/sanatorii/4_v_kompaniya_ooo
http://08.od.ua/turisticheskie_uslugi/sanatorii/4_v_kompaniya_ooo
http://08.od.ua/turisticheskie_uslugi/sanatorii/4_v_kompaniya_ooo
```

Later we'll want to aggregate records by web domain, and use the information in the individual page records to build features of the domains.

READY

Let's build a `traverse` function to output a lists of keys of a json record together with its tree depth and boolean is-leaf indicator:

```
%pyspark

from __future__ import print_function

def traverse(js, depth, keys):
    if type(js) is dict:
        d = depth + 1
        for k in js.keys():
            if type(js[k]) is not dict: leaf = 1
            else: leaf = 0
            keys += [(d,leaf,k)]
            depth, keys = traverse(js[k], d, keys)
    return depth, keys

js = sample[1]
depth, keys = traverse(js, 0, [])
keys.sort()

print(len(keys), "json keys:")
for x in keys:
    print(x)
```

READY

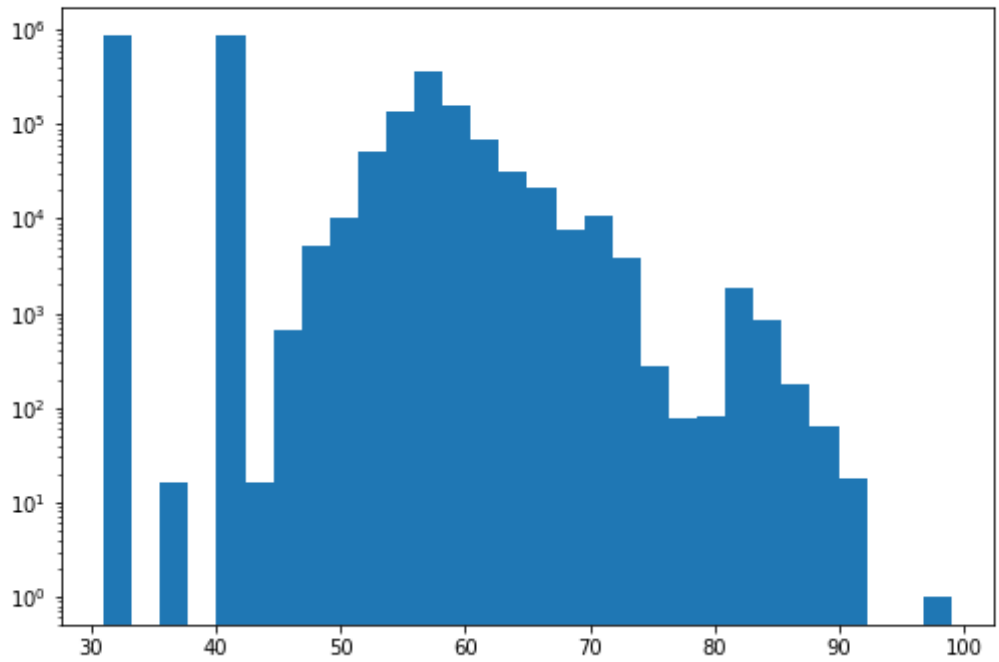
```
41 json keys:
(1, 0, u'Container')
(1, 0, u'Envelope')
(2, 0, u'Gzip-Metadata')
(2, 0, u'Payload-Metadata')
(2, 0, u'WARC-Header-Metadata')
(2, 1, u'Actual-Content-Length')
```

```
(2, 1, u'Block-Digest')
(2, 1, u'Compressed')
(2, 1, u'Filename')
(2, 1, u'Format')
(2, 1, u'Offset')
(2, 1, u'WARC-Header-Length')
(3, 0, u'HTTP-Request-Metadata')
(3, 1, u'Actual-Content-Type')
(3, 1, u'Content-Length')
(3, 1, u'Content-Type')
```

This json record has 41 keys. Let's see how that varies over all the records: READY

READY

```
%pyspark

from collections import Counter
import matplotlib.pyplot as plt

def get_json_keys(id_, iterator):

    for uri in iterator:

        file = unpack(uri)
        for record in file:
            if record['Content-Type'] == 'application/json':
                record = json.loads(record.payload.read())

                try:
                    _,k = traverse(record, 0, [])
                    """
                    yield the shape of the record:
                    """
                    yield dict(Counter([x[0:2] for x in k]))
                except KeyError:
                    yield None

json_keys = files.mapPartitionsWithIndex(get_json_keys)
json_shape = json_keys.collect()

total_shape = [sum(x.values()) for x in json_shape]

plt.hist(total_shape, bins=30)
plt.yscale('log')
plt.show()
```



Let's break down this histogram by tree depth and leaf vs node: READY

READY

```
%pyspark

from __future__ import print_function

maxdepth = max([k[0] for y in json_shape for k in y.keys()])

for depth in range(1, 1+maxdepth):
    print("Depth", depth)

    nodeshape = dict(Counter([x[(d,l)]  for x in json_shape for (d,l) in x.keys() if d==de
    if len(nodeshape.items()) > 0:
        print("nodes:")
        for i in nodeshape.items(): print(i)

    leafshape = dict(Counter([x[(d,l)]  for x in json_shape for (d,l) in x.keys() if d==de
    if len(leafshape.items()) > 0:
        print("leaves:")
        for i in leafshape.items(): print(i)
```

```
Depth 1
nodes:
(2, 2621614)
Depth 2
nodes:
(3, 2621614)
leaves:
(7, 2621614)
Depth 3
nodes:
(1, 2621614)
leaves:
(18, 438254)
```

```
(19, 435612)
(15, 1747748)
Depth 4
nodes:
```

In other words:

READY

At depth 1:
all records have two nodes

```
(1, 0, u'Container')
(1, 0, u'Envelope')
```

At depth 2:
all records have 3 nodes and 7 leaves

```
(2, 0, u'Gzip-Metadata')
(2, 0, u'Payload-Metadata')
(2, 0, u'WARC-Header-Metadata')
(2, 1, u'Actual-Content-Length')
(2, 1, u'Block-Digest')
(2, 1, u'Compressed')
(2, 1, u'Filename')
(2, 1, u'Format')
(2, 1, u'Offset')
(2, 1, u'WARC-Header-Length')
```

At depth 3:
all records a single 3 node

```
(3, 0, u'HTTP-Request-Metadata')
```

but 15, 18 or 19 leaves.

At depth 4:
2 or 3 nodes and 3,4,5 or 8 leaves.

At depth 5 and 6:

Depth 5 is where most of the leaf variance is; there's a single (optional) node

```
['Envelope']['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']
['Head']
```

with 1,2,3,4,5 leaves under it at depth 6.

Let's eyeball a record with a larger number of json keys:

READY

```
%pyspark

nkeys = 80

def f(rec):
    _,k = traverse(rec, 0, [])
```

```
    return len(k)

sample = files.\
        mapPartitionsWithSplit(json_mapper).\
        filter(lambda rec: f(rec) > nkeys).\
        take(100)

rec = sample[0]
depth, keys = traverse(rec, 0, [])
keys.sort()

print(len(keys))
for x in keys:
    print(x)
```

```
82
(1, 0, u'Container')
(1, 0, u'Envelope')
(2, 0, u'Gzip-Metadata')
(2, 0, u'Payload-Metadata')
(2, 0, u'WARC-Header-Metadata')
(2, 1, u'Actual-Content-Length')
(2, 1, u'Block-Digest')
(2, 1, u'Compressed')
(2, 1, u'Filename')
(2, 1, u'Format')
(2, 1, u'Offset')
(2, 1, u'WARC-Header-Length')
(3, 0, u'HTTP-Response-Metadata')
(3, 1, u'Actual-Content-Type')
(3, 1, u'Content-Length')
(3, 1, u'Content-Type')
(3, 1, u'Deflate-Length')
```

Let's look, at depth 5, at the `Head` node and a leaf `Links`, which contain relevant information about outgoing links:

READY

```
%pyspark

from pprint import pprint

node = rec['Envelope']['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']['Hea

_,k = traverse(node, 0, [])

for x in k:
    print(x)
    print(type(node[x[2]]))
    if type(node[x[2]]) == list: print(len(node[x[2]]))

pprint(node)
```

READY

```
(1, 1, u'Metas')
<type 'list'>
23
(1, 1, u'Link')
<type 'list'>
4
(1, 1, u'Scripts')
```

```
<type 'list'>
8
(1, 1, u'Title')
<type 'unicode'>
{u'Link': [{u'path': u'LINK@/href',
            u'rel': u'canonical',
            u'url': u'http://www.ardmoreite.com/article/20121123/OBITUARIES/121129904'},
           {u'path': u'LINK@/href',
            u'rel': u'stylesheet',
            u'type': u'text/css',
```

`Links` is a json leaf, but behaves like a node in that its value is a list of dicts: READY

```
%pyspark

node = rec['Envelope']['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']['Lin

_,k = traverse(node, 0, [])

for x in node2:
    pprint.pprint(x)
```

READY

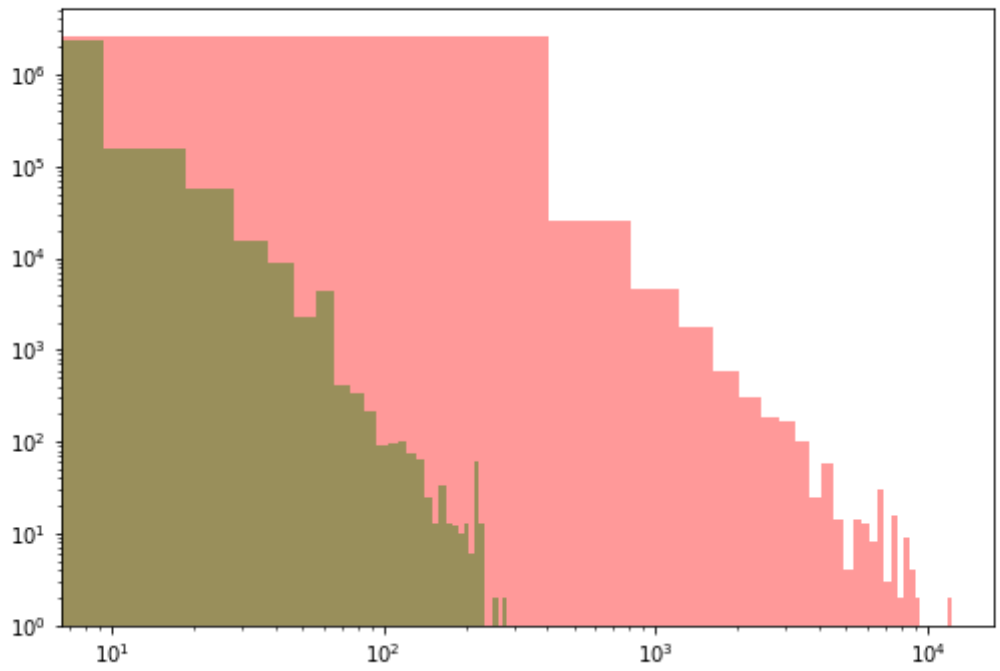
```
{u'path': u'IMG@/src',
 u'url': u'http://b.scorecardresearch.com/p?c1=2&c2=9289482&cv=2.0&cj=1'}
{u'path': u'A@/href', u'url': u'#'}
{u'path': u'A@/href',
 u'target': u'_top',
 u'text': u'News',
 u'url': u'http://www.ardmoreite.com/news'}
{u'path': u'A@/href',
 u'target': u'_top',
 u'text': u'Sports',
 u'url': u'http://www.ardmoreite.com/sports'}
{u'path': u'A@/href',
 u'target': u'_top',
 u'text': u'Entertainment',
 u'url': u'http://www.ardmoreite.com/entertainment'}
{u'path': u'A@/href',
 u'target': u'_top',
 u'text': u'Life',
```

Finally, compare the distributions (over json records) of the numbers of `'url'` values seen in a record, for the two nodes just discussed: READY

```
%pyspark

import matplotlib.pyplot as plt

def compare_links(record):
    try:
        set1 = set([x['url'] for x in record['Envelope']['Payload-Metadata']['HTTP-Respons
        set2 = set([x['url'] for x in record['Envelope']['Payload-Metadata']['HTTP-Respons
        return [len(set1), len(set2)]
    except KeyError:
```

READY

```
        return [0,0]

def get_link_counts(id_, iterator):
    for uri in iterator:
        file = unpack(uri)
        for record in file:
            if record['Content-Type'] == 'application/json':
                record = json.loads(record.payload.read())
                yield compare_links(record)

link_counts = files.mapPartitionsWithIndex(get_link_counts)
link_count = link_counts.collect()

llinks = [x[0] for x in link_count]
rlinks = [x[1] for x in link_count]

plt.hist(rlinks, bins=30, alpha=0.4, color='red')
plt.hist(llinks, bins=30, alpha=0.4, color='green')
plt.xscale('log')
plt.yscale('log')
```




```
%pyspark
```

READY