Bill 5 - constructin...

FINISHED

Constructing URI sets

What this notebook does:

Explores the URI paths under domains, introduces the idea of string signature and creates files of path sets in S3.

Took 1 sec. Last updated by anonymous at October 20 2017, 9:38:07 AM.

FINISHED

```
%pyspark

import boto
from boto.s3.key import Key
from gzipstream import GzipStreamFile
from pyspark.sql.types import *
import warc
import ujson as json
import urlparse

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

def unpack(uri):
    """
    Takes as argument one URI from
    watlist = sc.textFile("s3://commoncrawl/crawl-data/CC-MAIN-2017-04/wat.paths.gz")
    or WARC or WET file, and outputs the file for iterating over records.
    """
    conn = boto.connect_s3(anon=True, host='s3.amazonaws.com')
    bucket = conn.get_bucket('commoncrawl')
    key_ = Key(bucket, uri)
    file_ = warc.WARCFile(fileobj=GzipStreamFile(key_))
    return file_

def extract_json(id_, iterator):
    """
    Iterates through WARC records of an unpacked file in a Spark job.
    Usage:
    json_rdd = files.mapPartitionsWithIndex(extract_json)
    """
    for uri in iterator:
        file = unpack(uri)
        for record in file:
            if record['Content-Type'] == 'application/json':
                try:
                    content = json.loads(record.payload.read())
                    yield content['Envelope']
                except:
                    yield None

def parse_urls(record):
    """
```

```
    Takes WARC record and outputs all triples (domain, path, was_crawled) from URIs, if th
    It searches both target URI (was_crawled = 1) and out-links (was_crawled = 0).
    """
    url_list = []
    try:
        page_url = record['WARC-Header-Metadata']['WARC-Target-URI']
        x = urlparse.urlparse(page_url)
        if len(x.path) > 1:
            url_list += [(x.netloc, x.path, 1)]
    except:
        pass
    try:
        links = record['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']['Lin
        for url in links:
            x = urlparse.urlparse(url['url'])
            if len(x.path) > 1:
                url_list += [(x.netloc, x.path, 0)]
    except:
        pass

    return url_list
```

Took 0 sec. Last updated by anonymous at October 20 2017, 12:06:12 PM.

FINISHED

Parse URLs from JSON: domain records RDD

Each record is a domain name plus set of pairs URI + 'was_crawled' .

Took 0 sec. Last updated by anonymous at October 20 2017, 12:06:19 PM.

FINISHED

```
%pyspark

from __future__ import print_function

nfiles = 1
files = sc.parallelize(watlist.take(nfiles))

json_rdd = files.mapPartitionsWithIndex(extract_json)
json_rdd.cache()

print("Nr json records:", json_rdd.count())

domain_records = json_rdd\
        .flatMap(parse_urls)\
        .filter(lambda x: len(x[0]) > 0)\
        .map(lambda x: (x[0], (x[1], x[2])))\
        .groupByKey()\
        .map(lambda x: [x[0], set(list(x[1]))])

domain_records.cache()
json_rdd.unpersist()

domain_record_count = domain_records\
        .map(lambda x: (x[0], len(x[1]), max([y[1] for y in x[1]])))\
        .sortBy(lambda x: -x[1])\
        .collect()
for x in domain_record_count[:10]:
    print(x)
```

```
Nr json records: 162874
(u'www.facebook.com', 10870, 0)
(u'twitter.com', 10239, 0)
```

```
(u'www.newslocker.com', 5785, 1)
(u'artodyssey1.blogspot.com', 5367, 1)
(u'www.youtube.com', 5304, 1)
(u'plus.google.com', 4335, 1)
(u'www.socarrao.com.br', 3552, 1)
(u'4chanarchives.cu.cc', 3249, 1)
(u'www.price4all.ru', 3081, 1)
(u'akulagi.com', 3033, 1)
```

Took 3 min 5 sec. Last updated by anonymous at October 20 2017, 12:09:27 PM.

FINISHED

```
%pyspark

from __future__ import print_function

def depth(uri):
    return uri[:-1].count('/')

def length(uri):
    return len(uri) - uri.count('/')

def show_uri_pair(y):
    print(y[1], depth(y[0]), length(y[0]),  y[0])

ex = domain_records\
        .filter(lambda x: len(x[1]) > 10 and len(x[1]) < 100)\
        .takeSample(False, 10)

for dom in ex:
    print("-----------------------------------")
    print("Domain:", dom[0])
    print("Crawled:", max([y[1] for y in dom[1]]))
    print("Pages:")
    for y in sorted(dom[1]):
        show_uri_pair(y)
```

```
-----------------------------------
Domain: www.booksetc.co.uk
Crawled: 1
Pages:
0 1 4 /blog
0 3 23 /books/view/-9781785780233
1 3 64 /books/view/the-changing-british-party-system-1945-79-9780844733685
0 3 98 /features/view/1021-specialists-in-the-fields-of-eu-administration-politics-eu-selec
tion-competitions
0 3 79 /features/view/1117-education-nlp-coaching-hypnotherapy-self-help-from-crown-house
0 3 36 /features/view/117-moleskine-stationary
0 3 112 /features/view/1232-publishing-outstanding-works-of-knowledge-learning-research-for
-scholars-students-readers-world
0 3 70 /features/view/1346-emerald-group-publishing-linking-research-to-practice
0 3 92 /features/view/1483-usborne-children-s-books-leading-independent-children-s-publishe
r-in-the-uk
0 3 75 /features/view/1579-1579-a-wide-range-of-travel-language-products-from-berlitz
0 3 34 /features/view/1671-1671-quarto-books
```

Took 3 sec. Last updated by anonymous at October 20 2017, 12:44:32 PM.

Let's formalise these strings as signatures for the domains:

READY

```
```

%pyspark FINISHED

```
def domain_string(domain, path_set):
    """
    Takes domain and concatenates with path URIs separated by newlines.
    """
    out = domain + '\n' + '\n'.join(sorted([x[0] for x in list(path_set)])) + '\n\n\n'
    return out

ex = domain_records.takeSample(False, 1000)

for dom in ex:
    print("-----------------------------------")
```

```
-----------------------------------
sport.ay.by
/drugoe/
/inventar/
/ohota-i-rybolovstvo/
/trenazhery/
/turizm-i-otdyh/otdyh-tury/
/turizm-i-otdyh/turizm-aktivnyj-otdyh/
-----------------------------------
aw.zgamz.org
/play_3304.html
/play_8051.html
-----------------------------------
hitskin.com
/themes/10/16/57/i_up_arrow.gif
/themes/16/71/76/i_icon_mini_login.jpg
-----------------------------------
reviews.audioreview.com
```

Output exceeds 102400. Truncated.

Took 3 sec. Last updated by anonymous at October 20 2017, 12:36:07 PM.

The following count shows the motivation for encoding domains in this way. READY

We would like (for later use, when we model the string using an RNN) the alphabet of symbols in the representation to be reliably bounded. If we use the raw (unicode) string concatenation of the path URIs, then this is not the case because we get an explosion of possibilities from various languages. Here's a histogram of the symbols, together with their hex encodings:

%pyspark FINISHED

```
from collections import Counter

char_count = domain_records.map(lambda x: Counter('.'.join([y[0] for y in list(x[1])])))\
                    .aggregate(Counter(),
                            lambda acc, value: acc + value,
                            lambda acc1, acc2: acc1 + acc2)
char_count = dict(char_count)

def hexify(c):
    """
    Temporary ASCII encoding for human readable hex with ' - ' as delimiter for detecting
    non-Latin unicode.
    """
    try:
```

```
        s = c.encode("utf-8").encode("hex")
    except UnicodeDecodeError:
        s = 0
    n = len(s)
    if n <= 2: return s
    a = ' - '.join([s[i:i+2] for i in range(0,n,2)])
    return a[:-1]

# examine:
print("Nr characters:", len(char_count.keys()))
for key, value in sorted(char_count.iteritems(), key=lambda (k,v): (-v,k)):
    print "%8d %4s %16s" % (value, key, hexify(key))
```

```
('Nr characters:', 2083)
 5088043    /               2f
 4170303    e               65
 3711006    a               61
 3008168    -               2d
 2896274    t               74
 2800091    i               69
 2781947    s               73
 2723492    o               6f
 2456268    .               2e
 2448461    r               72
 2284189    n               6e
 2094450    l               6c
 1772045    c               63
 1644491    d               64
 1578650    m               6d
 1544678    p               70
 1485074    0               30
```

Took 16 sec. Last updated by anonymous at October 20 2017, 12:39:12 PM.

Compare this with the distribution after hexification. The number of symbols is bounded by 256 + 2 + 1. This time it's more informative to sort by key:

READY

```
%pyspark                                                                    FINISHED

from collections import Counter

hex_count = domain_records\
        .map(lambda x: [h for c in list(domain_string(x[0], x[1])) for h in hexify(c).spli
        .map(lambda x: Counter(x))\
        .aggregate(Counter(),
            lambda acc, value: acc + value,
            lambda acc1, acc2: acc1 + acc2)

hex_count = dict(hex_count)

# examine:
print("Nr hex characters:", len(hex_count.keys()))
for key, value in sorted(hex_count.iteritems(), key=lambda (k,v): k):
    print "%2s %8d" % (key, value)
```

```
('Nr hex characters:', 201)
 -   252198
09      407
0a  1991387
0b        1
0d      374
```

```
20    25008
21     1849
22       22
24     1285
25  1124638
26     2993
27      749
28     3563
29     3541
2a     2205
2b    31863
```

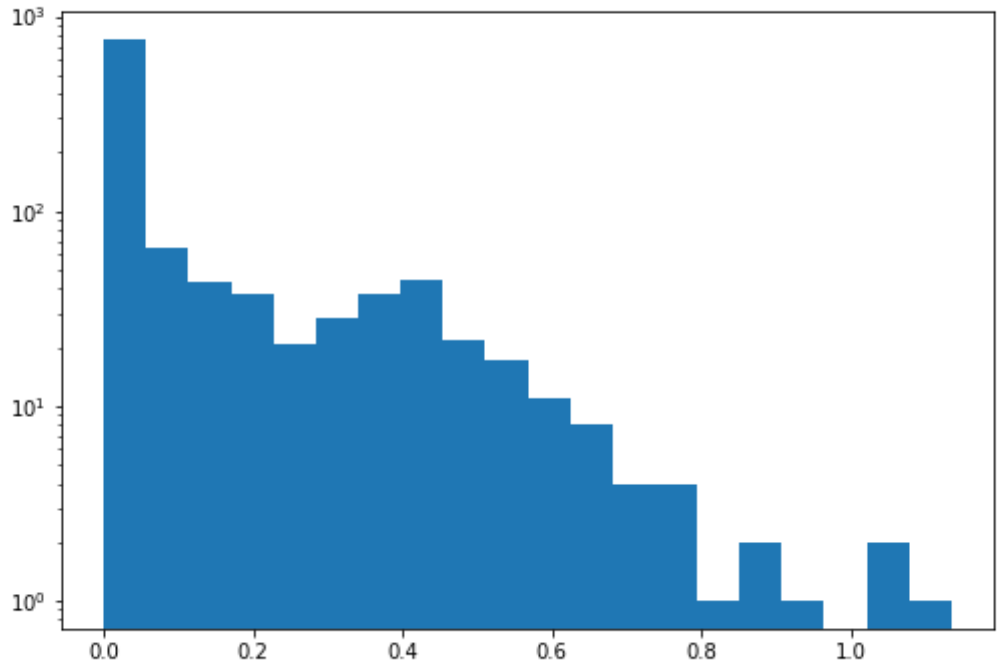
Took 1 min 11 sec. Last updated by anonymous at October 20 2017, 12:40:36 PM.

Let's use a filter on '-' to find all domains with non-Latin URIs. (This was the reason for the temporary notation.)

READY

```
%pyspark

import matplotlib.pyplot as plt

def nonlatin_detector(dom):
    """
    Computes the excess nr bytes over nr characters in a domain string.
    """
    str = domain_string(dom[0], dom[1])
    N = len(str)
    hex = [c.encode('utf-8').encode('hex') for c in list(str)]
    return float(sum([len(h)/2 for h in hex]) - N)/N

nonlatin = domain_records\
        .map(lambda x: (x[0], x[1], nonlatin_detector(x)))\
        .filter(lambda x: x[2] > 0)\
        .collect()

plt.hist([dom[2] for dom in nonlatin], bins=20)
plt.yscale("log")
plt.show()
```

FINISHED



Took 48 sec. Last updated by anonymous at October 20 2017, 12:41:35 PM.

For example: READY

```
%pyspark

nonlatin[1]
```

FINISHED

```
(u'bahriatowntoday.com', set([(u'/category/jobs/', 0), (u'/wp-content/uploads/2014/11/04/Ba
hria-Town-Karachi-Bahria-Apartments.jpg', 0), (u'/wp-content/uploads/2014/11/04/Bahria-Town
-Karachi-200-Square-Yards-Bahria-Homes.jpg', 0), (u'/wp-content/uploads/2014/11/04/Bahria-T
own-Karachi-Gate-House.jpg', 0), (u'/wp-content/themes/wp-clear31/images/facebook.png', 0),
 (u'/happy-new-2017/', 0), (u'/bahria-town-new-year-celebrations-2017/', 0), (u'/fbr-proper
ty-valuation-table-tax-of-bahria-town-karachi/', 0), (u'/precincts-of-bahria-town-karach
i/', 0), (u'/a-birds-eye-view-of-bahria-farmhouses-karachi-december-2016/', 0), (u'/wp-cont
ent/uploads/2014/01/09/Bahria-Town-Islands-City-Karachi-Logo.jpg', 0), (u'/wp-content/uploa
ds/2015/10/06/High-Resolution-Precincts-Maps-of-Bahria-Town-Karachi.jpg', 0), (u'/bahria-sp
orts-city-karachi-precinct-44-high-resolution-map/', 0), (u'/render-200-square-yards-bahria
-homes-in-bahria-town-karachi/', 0), (u'/wp-content/uploads/2014/12/14/Bahria-Golf-City-Bah
ria-Town-Karachi-Banner.jpg', 0), (u'/wp-content/uploads/2014/11/04/Bahria-Town-Karachi-125
-Square-Yards-Bahria-Homes.jpg', 0), (u'/wp-content/uploads/2013/10/11/Bahria-Town-Tower.jp
g', 0), (u'/wp-content/uploads/2013/10/04/Bahria-Town-Today-Logo-01.png', 0), (u'/pakistans
-first-ever-36-hole-pga-standard-night-lit-golfing-facility-in-bahria-golf-city-karachi/',
 0), (u'/bahria-sports-city-karachi-high-resolution-master-plan-location-map/', 0), (u'/ta
g/Hoshang-Pearl', 0), (u'/bahria-town-karachi-latest-progress-update-october-2016/', 0),
 (u'/tag/Bahria-Town-Karachi-Overseas-Block/', 0), (u'/category/events/', 0), (u'/bahria-sp
```

Took 0 sec. Last updated by anonymous at October 20 2017, 12:42:46 PM.

```
%pyspark

from __future__ import print_function

for dom in nonlatin:
```

FINISHED

```
    if dom[2] > 0.05:
        print("-----------------------------------------")
        print("%s (%g)" % (dom[0], dom[2]))
        for uri in sorted(dom[1]):
            show_uri_pair(uri)
```

```
----------------------------------------
hf5.l2central.info (0.508772)
0 1 34 /Категория:Земли Иннадрила - квесты
----------------------------------------
iau.vec.go.th (0.13914)
0 1 59 /%E0%B8%81%E0%B8%8E%E0%B8%9A%E0%B8%B1%E0%B8%95%E0%B8%A3.aspx
0 1 158 /%E0%B8%82%E0%B9%88%E0%B8%B2%E0%B8%A7%E0%B8%9B%E0%B8%A3%E0%B8%B0%E0%B8%8A%E0%B8%B2%
E0%B8%AA%E0%B8%B1%E0%B8%A1%E0%B8%9E%E0%B8%B1%E0%B8%99%E0%B8%98%E0%B9%8C.aspx
0 2 203 /%E0%B8%94%E0%B8%B2%E0%B8%A7%E0%B8%99%E0%B9%8C%E0%B9%82%E0%B8%AB%E0%B8%A5%E0%B8%9
4/%E0%B8%81%E0%B8%B2%E0%B8%A3%E0%B8%AD%E0%B8%9A%E0%B8%A3%E0%B8%A1%E0%B8%AA%E0%B8%B1%E0%B8%A
1%E0%B8%A1%E0%B8%99%E0%B8%B2.aspx
0 2 140 /%E0%B8%94%E0%B8%B2%E0%B8%A7%E0%B8%99%E0%B9%8C%E0%B9%82%E0%B8%AB%E0%B8%A5%E0%B8%9
4/%E0%B8%84%E0%B8%B9%E0%B9%88%E0%B8%A1%E0%B8%B7%E0%B8%AD.aspx
0 2 194 /%E0%B8%94%E0%B8%B2%E0%B8%A7%E0%B8%99%E0%B9%8C%E0%B9%82%E0%B8%AB%E0%B8%A5%E0%B8%9
4/%E0%B8%AB%E0%B8%99%E0%B8%B1%E0%B8%87%E0%B8%AA%E0%B8%B7%E0%B8%AD%E0%B9%80%E0%B8%A7%E0%B8%B
5%E0%B8%A2%E0%B8%99.aspx
0 1 86 /%E0%B8%95%E0%B8%B4%E0%B8%94%E0%B8%95%E0%B9%88%E0%B8%AD%E0%B9%80%E0%B8%A3%E0%B8%B2.a
spx

Output exceeds 102400. Truncated.
```

Took 12 sec. Last updated by anonymous at October 20 2017, 12:45:08 PM.

```
%pyspark

domain_records.unpersist()
```

FINISHED

```
PythonRDD[242] at RDD at PythonRDD.scala:48
```

Took 0 sec. Last updated by anonymous at October 20 2017, 12:45:51 PM.

READY

Save to S3

The end-to-end process:

READY

```
%pyspark

nfiles = 1024

files = sc.parallelize(watlist.take(nfiles))
json_rdd = files.mapPartitionsWithIndex(extract_json)

domains_rdd = json_rdd\
        .flatMap(parse_urls)\
        .filter(lambda x: len(x[0]) > 0)\
        .map(lambda x: (x[0], (x[1], x[2])))\
        .groupByKey()\
        .map(lambda x: {'domain': x[0], 'path_set': set(x[1])})

# make sure the following S3 directory doesn't already exist:
```

FINISHED

```
outputURI = "s3://billsdata.net/CommonCrawl/domain_paths_from_%d_WAT_files" % nfiles
codec = "org.apache.hadoop.io.compress.GzipCodec"
domains_rdd.saveAsTextFile(outputURI, codec)
```

Took 1 hrs 32 min 20 sec. Last updated by anonymous at October 20 2017, 3:03:59 PM.

Timings: READY

Cluster	nr WAT files	time	output size (gzip)
16 x m4.2xlarge	128	7 min 24 sec	944.6 MiB
16 x m4.2xlarge	256	10 min 16 sec	1.7 GiB
16 x m4.2xlarge	512	19 min 31 sec	3.1 GiB
16 x m3.xlarge	1024	1 hrs 44 min 23 sec	5.7 GiB
16 x m4.2xlarge	1024	40 min 43 sec	5.7 GiB

To find output size:

```
$ aws s3 ls —human-readable —summarize
s3://billsdata.net/CommonCrawl/domain_paths_from_256_WAT_files/ | grep Total
```

```
%pyspark
```

READY