Bill 5 - understandi...

```
%pyspark
```

FINISHED

```
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):
    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):
    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):
    url_list = []
    try:
        page_url = record['WARC-Header-Metadata']['WARC-Target-URI']
        x = urlparse.urlparse(page_url)
        url_list += [(x.netloc, x.path)]
    except:
        pass
    try:
        links = record['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']['Lin
        for url in links:
            x = urlparse.urlparse(url['url'])
            url_list += [(x.netloc, x.path)]
    except:
        pass

    return url_list
```

Took 44 sec. Last updated by anonymous at September 09 2017, 7:10:06 AM.

FINISHED

Parse URLs from JSON: Records RDD

Took 0 sec. Last updated by anonymous at September 09 2017, 7:26:40 AM.

READY

```
%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())

records = json_rdd\
        .flatMap(parse_urls)\
        .filter(lambda x: x[0] is not "")\
        .groupByKey()\
        .map(lambda x: (x[0], set(x[1])))

records.cache()
json_rdd.unpersist()

record_count = records.map(lambda x: (x[0], len(x[1]))).sortBy(lambda x: -x[1]).collect()
for x in record_count[:10]: print(x)
```

```
Nr json records: 162874
(u'www.facebook.com', 10872)
(u'twitter.com', 10241)
(u'www.newslocker.com', 5784)
(u'artodyssey1.blogspot.com', 5366)
(u'www.youtube.com', 5305)
(u'plus.google.com', 4337)
(u'www.socarrao.com.br', 3551)
(u'4chanarchives.cu.cc', 3249)
(u'www.price4all.ru', 3079)
(u'akulagi.com', 3034)
```

READY

```
%pyspark

from __future__ import print_function

ex = records.filter(lambda x: len(x[1])==10).takeSample(False,1)[0]
print("Domain:", ex[0])
print("Pages:")
for y in ex[1]: print(y)
```

```
Domain: pi.lmcdn.ru
Pages:
/img600x866/L/I/LI024LWHGS69_2_v1.jpg
/img600x866/L/I/LI024LWHGS65_2_v1.jpg
/img600x866/B/E/BE007GWDSQ97_2_v1.jpg
/img600x866/A/D/AD094CWFSM34_2_v1.jpg
/img600x866/L/I/LI024LWHGS68_2_v1.jpg
/img600x866/B/E/BE007GWDSQ97_1_v1.jpg
/img600x866/A/D/AD094CWFSM34_3_v1.jpg
/img600x866/A/D/AD094CWFSM34_1_v1.jpg
/img600x866/L/I/LI024LWHGS66_2_v1.jpg
/img600x866/A/D/AD094CWFSM34_4_v1.jpg
```

READY

We next define a string encoding of domains.
The idea will be to choose this so that domain structure (as contained in its URIs) can be learnt be an RNN.

```
%pyspark
```
FINISHED

```
import re
from __future__ import print_function

def hexify(c):
    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]

def hexalise(str):
    return ' '.join([hexify(c) for c in str]) + ' . '

def domain_string(domain, path_set):
    out = hexalise(domain)
    for p in path_set: out += hexalise(p)
    return out
```

Took 0 sec. Last updated by anonymous at September 09 2017, 7:12:35 AM.

As the examples below show, we've chosen this encoding with the following constraints in mind: READY

- All symbols should be separated by spaces in order to parse at RNN training time.
- As well as hex symbols we include '.' to delimit different URIs.
- We include '-' as a limiter within non-Latin unicode characters. This will allow the RNN to distinguish Chinese characters, say, from sequences of Latin characters.
- Distinct domains will be delimited by '\n' at RNN training time.

```
%pyspark
```
READY

```
from __future__ import print_function

ex = 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("Page string:")
    print('|'.join(list(dom[1])))
```

```
s-sobre/setores/|/pme/delivery-recebe-pedido-inusitado-e-da-uma-aula-de-bom-atendimento/|/c
arreira/quer-escrever-bem-nao-tente-parecer-inteligente/|/./mercados/|/./economia/|/noticia
s-sobre/airbnb/|/expediente|/./marketing/|/topicos/carros|/./revista-exame/|/noticias-sobr
e/economia-colaborativa/|/noticias-sobre/revista-voce-sa/|/./brasil/|/noticias-sobre/edicao
-199/|/noticias-sobre/redes-sociais/|/topicos/roupas|/./ciencia/|/./seu-dinheiro/|/rss/|/./
tecnologia/|/pme/conheca-a-fabrica-de-bronzeadas-que-esta-fazendo-sucesso-no-rio/|/politica
-de-privacidade|/noticias-sobre/prisoes/|/carreira/as-50-empresas-mais-amadas-pelos-seus-fu
ncionarios-no-brasil/|/noticias-sobre/bens-de-consumo/|/termos-de-uso|/./estilo-de-vida/|/n
egocios/a-glamorosa-vida-do-criador-do-snapchat-evan-spiegel/|/noticias-sobre/internet/|/no
```

```
ticias-sobre/desemprego
-----------------------------------------
Domain: www.aquaristikshop.com
Page string:
/cgi-bin/neu/webshop.pl|/aquaristik/EHEIM-InstallationsSET-1/400430/|/aquaristik/EHEIM-Prof
ildichtung-professionel-eXperience/734315/|/aquaristic/Tetra-Pond-Koi-Sticks/1105020/|/asse
ts/images/right_s.gif|/aquaristik/historie|/aquaristik/Tropic-Marin-Pro-Reef-Meersalz/30707
0/|/aquaristik/gartenteich/teichfutter/|/de/|/aquaristik/EHEIM-Filtervlies-fuer-professione
l-und-eXperience/2616265/|/aquaristik/Mag-Float-Algenmagnet-schwimmend-lang/278003/|/aquari
```

```
%pyspark                                                                    READY

ex = records.filter(lambda x: len(x[1])==10).take(2)

for dom in ex:
    print("-----------------------------------------")
    print("Domain:", dom[0])
    print("Page string:")
    print(domain_string(dom[0], dom[1]))
```

```
 67 73 72 65 63 68 74 2e 70 68 70 . 2f 4d 6f 64 75 6c 65 2f 56 65 72 6b 65 68 72 73 7a 69 7
6 69 6c 72 65 63 68 74 2e 70 68 70 . 2f 54 65 78 74 65 2f 52 73 70 72 32 31 38 37 2e 70 68
 70 . 2f 4d 6f 64 75 6c 65 2f 56 65 72 6b 65 68 72 73 73 74 72 61 66 73 61 63 68 65 6e 2e 7
0 68 70 . 2f 4c 65 78 69 6b 6f 6e 2e 70 68 70 . 2f 49 6d 70 72 65 73 73 75 6d 2e 70 68 70 .

-----------------------------------------
('Domain:', u'www.charityblossom.org')
Page string:
77 77 77 2e 63 68 61 72 69 74 79 62 6c 6f 73 73 6f 6d 2e 6f 72 67 .  . 2f 64 69 72 65 63 74
 6f 72 79 2f 46 4c 2f 4f 72 6c 61 6e 64 6f 2f 33 32 38 31 31 2f . 2f 64 69 72 65 63 74 6f 7
2 79 2f 4b 53 2f 54 6f 77 61 6e 64 61 2f 63 61 74 65 67 6f 72 79 2f 70 75 62 6c 69 63 2d 73
 61 66 65 74 79 2d 64 69 73 61 73 74 65 72 2d 70 72 65 70 61 72 65 64 6e 65 73 73 2d 72 65
 6c 69 65 66 2d 6d 2f 6d 61 6e 61 67 65 6d 65 6e 74 2d 74 65 63 68 6e 69 63 61 6c 2d 61 73
 73 69 73 74 61 6e 63 65 2d 6d 30 32 2f . 2f 6e 6f 6e 70 72 6f 66 69 74 2f 61 6d 65 72 69 6
3 61 6e 2d 6c 65 67 69 6f 6e 2d 64 75 6e 6b 69 72 6b 2d 6e 79 2d 31 34 30 34 38 2d 65 64 6d
 75 6e 64 2d 66 2d 67 6f 75 6c 64 2d 6a 72 2d 31 36 30 37 32 30 31 36 33 2f . 2f 6e 6f 6e 7
0 72 6f 66 69 74 2f 74 65 63 68 6e 6f 6c 6f 67 79 2d 72 65 76 69 65 77 2d 69 6e 63 2d 63 61
 6d 62 72 69 64 67 65 2d 6d 61 2d 30 32 31 34 32 2d 6a 61 6d 65 73 2d 63 6f 79 6c 65 2d 39
```

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                                                                    READY

from collections import Counter

char_count = records.map(lambda x: Counter('.'.join(list(x[1]))))\
                    .aggregate(Counter(),
                           lambda acc, value: acc + value,
                           lambda acc1, acc2: acc1 + acc2)
char_count = dict(char_count)
```

```
# examine:
print("Nr characters:", len(char_count.keys()))
for key, value in sorted(char_count.iteritems(), key=lambda (k,v): (-v,k)):
```

```
     62    区    e5 - 8c - ba
     61    ﾚ    e0 - b9 - 84
     59    ﾞ    e0 - b9 - 87
     55    ◌    e0 - af - 8d
     55    品    e5 - 93 - 81
     54    ß         c3 - 9f
     54    ş         c5 - 9f
     54    と    e3 - 81 - a8
     53    Ø         d0 - a4
     52    寶    e5 - af - b6
     50    σ         cf - 83
     50    ナ    e3 - 83 - 8a
     49    生    e7 - 94 - 9f
     48    新    e6 - 96 - b0
     47    ñ         c3 - b1
     47    _         d9 - 80
     47    ラ    e3 - 83 - a9
     46    ύ         cf - 8d
```

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

READY

```
%pyspark

from collections import Counter

hex_count = records.map(lambda x: Counter(domain_string(x[0], x[1]).split()))\
                    .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:', 199)
 -   252648
 .  1950605
03        1
09      413
0a      573
0b        1
0d      414
20    25473
21     1845
22       23
24     1291
25  1122548
26     3063
27      750
28     3561
29     3541
```

```
%pyspark

records.unpersist()
```

READY

```
PythonRDD[52] at RDD at PythonRDD.scala:48
```

FINISHED

Save to S3

Took 0 sec. Last updated by anonymous at September 09 2017, 7:28:01 AM.

The end-to-end process:

READY

```
%pyspark

nfiles = 128

files = sc.parallelize(watlist.take(nfiles))
json_rdd = files.mapPartitionsWithIndex(extract_json)
domains_rdd = json_rdd\
            .flatMap(parse_urls)\
            .filter(lambda x: x[0] is not "")\
            .groupByKey()\
            .map(lambda x: {'domain': x[0], 'path_set': set(x[1])})


# make sure the following S3 directory is deleted first:

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

READY

Timings:

FINISHED

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 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_signatures_256_WAT_files/ | grep Total
```

Took 0 sec. Last updated by anonymous at September 09 2017, 9:31:44 AM.

```
%pyspark
```

READY