201708 common c...

READY

Building domain features from WAT

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):
    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 30 sec. Last updated by anonymous at September 07 2017, 5:36:46 PM.

```
%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)
```

FINISHED

```
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)
```

Took 2 min 59 sec. Last updated by anonymous at September 07 2017, 8:42:43 PM.

```
%pyspark

from __future__ import print_function

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

FINISHED

```
Domain: www.missme.com
Pages:
/accessories/view-all/
/accessories/belts/
/
/jeans/view-all/
/outerwear/view-all/
/girls/view-all/
/giftcertificates.php
/mm-couture/view-all/
/bottoms/view-all/
/tops/view-all/
```

Took 1 sec. Last updated by anonymous at September 07 2017, 8:42:50 PM.

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.

FINISHED

Took 0 sec. Last updated by anonymous at September 07 2017, 9:29:49 PM.

```
%pyspark

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 normalise(s):
    return ' '.join([hexify(c) for c in s]) + ' . '

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

FINISHED

Took 0 sec. Last updated by anonymous at September 07 2017, 8:45:58 PM.

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

- 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.

FINISHED

Took 0 sec. Last updated by anonymous at September 07 2017, 9:34:13 PM.

```
%pyspark

from __future__ import print_function

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

for dom in ex:
    print("-----------------------------------------")
    print("Domain:", dom[0])
    print("Page string:")
    print('|'.join(list(dom[1])))
```

FINISHED

```
999-55670505829.jpg|/images/Thumbnails/swa/tch/999999-55670505768_swatch.jpg|/images/Large/
910/726/999999-41457910726.jpg|/images/Thumbnails/634/_sw/999999-814714026634_sw.jpg|/image
s/Thumbnails/059/536/999999-75020059536.jpg|/images/Thumbnails/222/0_1/425991-37988562220_
1.jpg|/images/Thumbnails/503/221/999999-55670503221.jpg|/images/Large/910/764/999999-414579
10764.jpg|/images/Thumbnails/904/993/999999-6845904993.jpg|/images/Thumbnails/474/739/47473
9.jpg|/images/Large/533/3_3/999999-7713036555333_3.jpg|/images/Large/533/3_4/999999-7713036
555333_4.jpg|/images/Thumbnails/046/413/999999-75020046413.jpg|/images/Thumbnails/214/544/6
```

```
000188214544.jpg|/images/Thumbnails/909/837/425991-64547909837.jpg|/images/Large/533/3_1/99
9999-7713036555333_1.jpg|/images/Thumbnails/512/151/1512151.jpg|/images/Large/522/543/99999
9-55670522543.jpg|/images/Thumbnails/swa/tch/28995028960_swatch.jpg|/images/Thumbnails/353/
742/1353742.jpg|/images/Large/120/272/999999-722674120272.jpg|/images/Thumbnails/641/_sw/99
9999-814714026641_sw.jpg|/images/Thumbnails/672/054/6000190672054.jpg|/images/Thumbnails/63
8/869/6000192638869.jpg|/images/Large/638/869/6000192638869.jpg|/images/Large/202/5_1/99999
9-64132032025_1.jpg|/images/Large/202/5_2/999999-64132032025_2.jpg|/images/Thumbnails/swa/t
ch/999999-55670505775_swatch.jpg|/images/Thumbnails/505/836/999999-55670505836.jpg|/images/
Thumbnails/swa/tch/999999-55670505782_swatch.jpg|/images/Thumbnails/swa/tch/999999-55670505
829_swatch.jpg|/images/Thumbnails/505/775/999999-55670505775.jpg|/images/Large/533/3_2/9999
```

Output exceeds 102400. Truncated.

Took 1 sec. Last updated by anonymous at September 07 2017, 9:50:52 PM.

```
%pyspark

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]))
```

FINISHED

```
-----------------------------------------
('Domain:', u'www.missme.com')
Page string:
77 77 77 2e 6d 69 73 73 6d 65 2e 63 6f 6d . 2f 61 63 63 65 73 73 6f 72 69 65 73 2f 76 69 65
 77 2d 61 6c 6c 2f . 2f 61 63 63 65 73 73 6f 72 69 65 73 2f 62 65 6c 74 73 2f . 2f . 2f 6a
 65 61 6e 73 2f 76 69 65 77 2d 61 6c 6c 2f . 2f 6f 75 74 65 72 77 65 61 72 2f 76 69 65 77 2
d 61 6c 6c 2f . 2f 67 69 72 6c 73 2f 76 69 65 77 2d 61 6c 6c 2f . 2f 67 69 66 74 63 65 72 7
4 69 66 69 63 61 74 65 73 2e 70 68 70 . 2f 6d 6d 2d 63 6f 75 74 75 72 65 2f 76 69 65 77 2d
 61 6c 6c 2f . 2f 62 6f 74 74 6f 6d 73 2f 76 69 65 77 2d 61 6c 6c 2f . 2f 74 6f 70 73 2f 76
 69 65 77 2d 61 6c 6c 2f .
-----------------------------------------
('Domain:', u'healthland.time.com')
Page string:
68 65 61 6c 74 68 6c 61 6e 64 2e 74 69 6d 65 2e 63 6f 6d . 2f 32 30 31 32 2f 31 30 2f 30 38
 2f 36 2d 62 72 65 61 74 68 69 6e 67 2d 65 78 65 72 63 69 73 65 73 2d 74 6f 2d 72 65 6c 61
 78 2d 69 6e 2d 31 30 2d 6d 69 6e 75 74 65 73 2d 6f 72 2d 6c 65 73 73 2f . 2f 61 75 74 68 6
f 72 2f 62 72 6f 63 68 6d 61 6e 2f . 2f 32 30 31 31 2f 30 35 2f 30 35 2f 74 72 75 65 2d 6f
```

Took 0 sec. Last updated by anonymous at September 07 2017, 8:46:00 PM.

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

FINISHED

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:

Took 0 sec. Last updated by anonymous at September 07 2017, 9:34:34 PM.

```
%pyspark

from collections import Counter
```

FINISHED

```
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)):
    print "%8d %4s %16s" % (value, key, hexify(key))
```

```
       1    열    ec - 97 - b4
       1    용    ec - 9a - a9
       1    운    ec - 9a - b4
       1    웃    ec - 9b - 83
       1    워    ec - 9b - 8c
       1    원    ec - 9b - 90
       1    웨    ec - 9b - a8
       1    유    ec - 9c - a0
       1    인    ec - 9d - b8
       1    정    ec - a0 - 95
       1    제    ec - a0 - 9c
       1    조    ec - a1 - b0
       1    주    ec - a3 - bc
       1    지    ec - a7 - 80
       1    차    ec - b0 - a8
       1    축    ec - b6 - 95
       1    츠    ec - b8 - a0
       1    카    ec - b9 - b4
```

Took 0 sec. Last updated by anonymous at September 07 2017, 9:20:20 PM. (outdated)

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:

FINISHED

Took 0 sec. Last updated by anonymous at September 07 2017, 9:24:09 PM.

```
%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)
```

FINISHED

```
90      419
91      326
92      326
93      446
94      680
95      646
96      377
97      516
98      541
99      969
9a      770
```

```
9a      770
9b      660
9c      493
9d      306
9e      385
9f      717
a0      430
```

Took 27 sec. Last updated by anonymous at September 07 2017, 9:18:21 PM.

```
%pyspark

records.unpersist()
```

FINISHED

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

Took 0 sec. Last updated by anonymous at September 07 2017, 8:02:38 PM.

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: 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_signatures_%d_WAT_files" % nfiles
codec = "org.apache.hadoop.io.compress.GzipCodec"
domains_rdd.saveAsTextFile(outputURI, codec)
```

READY

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 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
```

Next read the data from this bucket:

```
%pyspark

from __future__ import print_function

nfiles = 128
inputURI = "s3://billsdata.net/CommonCrawl/domain_signatures_%d_WAT_files/" % nfiles

domains_rdd = sc.textFile(inputURI).map(eval)
domains_rdd.cache()

domain_uri_count = domains_rdd\
                    .map(lambda x: [len(x['path_set']), sum([len(uri) for uri in x['path_s
                    .aggregate((0, 0, 0),
                               lambda acc, value: (acc[0] + 1, acc[1] + value[0], acc[2] +
                               lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1], (

print("Nr domains: %15d" % domain_uri_count[0])
print("Nr page URIs: %13d" % domain_uri_count[1])
print("Nr URI chars: %13d" % domain_uri_count[2])
```

FINISHED

```
Nr domains:         2626203
Nr page URIs:      71799497
Nr URI chars:    3259974688
```

Took 1 min 8 sec. Last updated by anonymous at September 07 2017, 8:11:10 PM.

```
%pyspark

domains_rdd.unpersist()
```

FINISHED

```
PythonRDD[73] at RDD at PythonRDD.scala:48
```

Took 0 sec. Last updated by anonymous at September 07 2017, 8:38:41 PM.

Write to local HDFS a single string for all domains:

READY

```
%pyspark


big_domain_string = domains_rdd\
    .map(lambda x: domain_string(x['domain'], x['path_set']))

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

FINISHED

Took 11 min 40 sec. Last updated by anonymous at September 07 2017, 8:22:57 PM.

To concatenate into a single gzip file:

FINISHED

```
$ aws s3 sync
s3://billsdata.net/CommonCrawl/domain_hex_strings_1024_WAT_files/ ./tmp
$ gunzip -c ./tmp/part*.gz | cat | gzip -c > ./tmp/big_domain_string_1024.gz
$ aws s3 sync big_domain_string_1024.gz s3://billsdata.net/CommonCrawl/
```

Took 0 sec. Last updated by anonymous at September 07 2017, 8:24:51 PM.

```
%pyspark

nfiles = 128

path_inputURI = "s3://billsdata.net/CommonCrawl/domain_signatures_%d_WAT_files/" % nfiles
hex_inputURI = "s3://billsdata.net/CommonCrawl/domain_hex_strings_%d_WAT_files/" % nfiles
domain_paths = sc.textFile(path_inputURI)
domain_paths.cache()
domain_strings = sc.textFile(hex_inputURI)
domain_strings.cache()

print(domain_paths.count())
print(domain_strings.count())
```

FINISHED

```
2626203
2626203
```

Took 1 min 30 sec. Last updated by anonymous at September 07 2017, 8:32:08 PM.

```
%pyspark

domain_paths.unpersist()
domain_strings.unpersist()
```

FINISHED

```
s3://billsdata.net/CommonCrawl/domain_hex_strings_128_WAT_files/ MapPartitionsRDD[81] at te
xtFile at NativeMethodAccessorImpl.java:0
```

Took 0 sec. Last updated by anonymous at September 07 2017, 8:39:12 PM.

Compute the distribution of characters for a small sample:

FINISHED

Took 0 sec. Last updated by anonymous at September 07 2017, 5:38:23 PM.

```
%pyspark

len(domain_strings.take(1)[0])
```

FINISHED

```
890047
```

Took 0 sec. Last updated by anonymous at September 07 2017, 8:36:33 PM.

```
%pyspark

from collections import Counter

big_path_sample = domain_paths.take(2000)

# either:
"""
char_count = sc.parallelize(big_string_sample)\
        .flatMap(lambda s: Counter(s).items())\
        .reduceByKey(lambda x,y: x+y)\
        .collect()
"""
# or:
char_count = sc.parallelize(big_string_sample)\
        .map(lambda s: Counter(s))\
        .aggregate(Counter(),
                    lambda acc, value: acc + value,
                    lambda acc1, acc2: acc1 + acc2)
```

FINISHED

```
# convert to dict:
char_count = dict(char_count)


# examine:
print("Nr characters", len(char_count.keys()))
print("Counts (unicode, ASCII, hex):")
for key, value in sorted(char_count.iteritems(), key=lambda (k,v): (-v,k)):
    print "%s %20s %8d" % (key,
                           #key.encode("iso-8859-1", "replace"),
                           hexify(key),
                           value)
```

```
        ef - bf - bd        32
士       e5 - a3 - ab        30
林       e6 - 9e - 97        30
雅       e9 - 9b - 85        30
م            d9 - 85        29
т            d1 - 82        25
ˆ            cb - 86        24
и            d0 - b8        23
†       e2 - 80 - a0        23
ت            d8 - aa        22
{                 7b        20
c            d1 - 81        20
ي            d9 - 8a        20
ز            d8 - b2        19
ه            da - be        19
试       e8 - af - 95        19
验       e9 - aa - 8c        19
e            d0 - b5        18
```

Took 1 sec. Last updated by anonymous at September 07 2017, 5:56:09 PM. (outdated)

```
%pyspark
```

FINISHED

```
c = 'ب'
d = '第'


def hexify(c):
    s = c.encode("hex")
    if len(s) <= 2: return s
    a = ' '.join([s[i:i+2]+' -' for i in range(0,len(s),2)])
    return a[:-1]

print(hexify(c))
print(hexify(d))
```

```
d8 - a8
e7 - ac - ac
```

Took 0 sec. Last updated by anonymous at September 07 2017, 5:52:33 PM.

Now let's look at basic statistics of the path URI for a domain…

READY

```
%pyspark
```

READY

```
import re
```

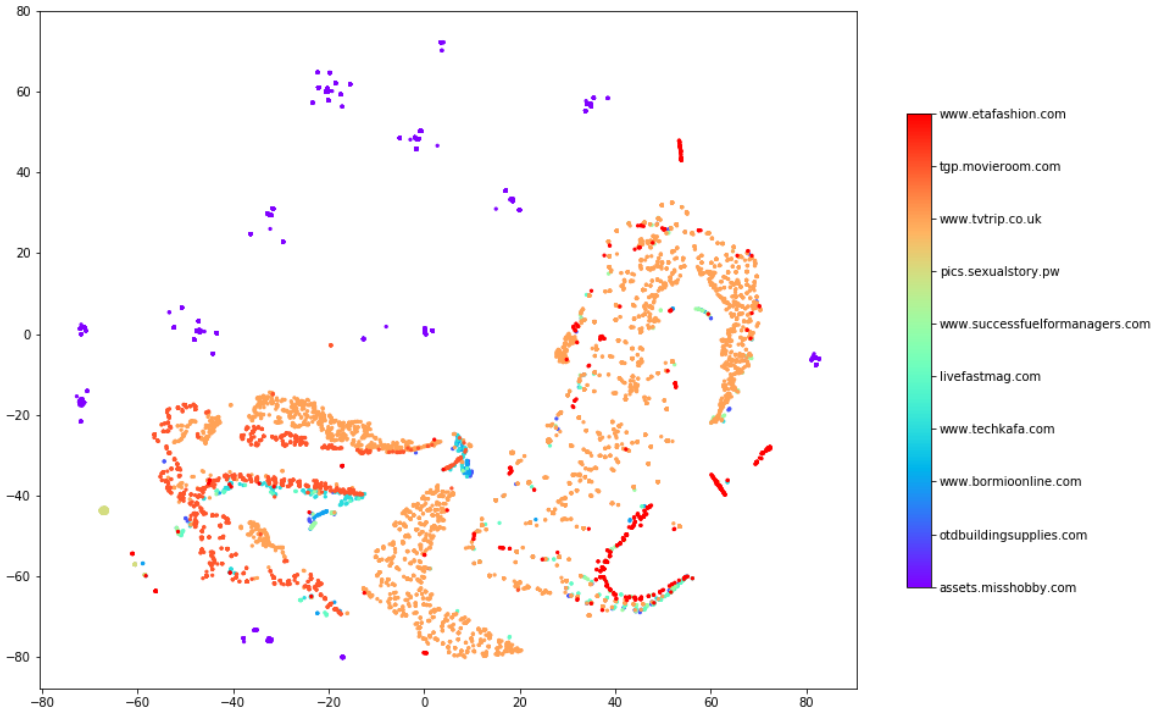
```
from math import log

def string_features(str):
    N = float(len(str))
    if N==0: return None
    a = len(re.findall(r'/', str))/N
    b = len(re.findall(r'\.', str))/N
    c = len(re.findall(r'-', str))/N
    d = len(re.findall(r'_', str))/N
    cap = len(re.findall(r'[A-Z]', str))/N
    num = len(re.findall(r'[0-9]', str))/N
    """
    PLUS: evaluate through a trained RNN for additional vector representation
    """
    return [log(N), a, b, c, d, num, cap]

def domain_features(domain, path_set):
```

```
%pyspark                                                                    READY

page_feature_rdd = domains_rdd\
                    .map(lambda x: (x['domain'], [string_features(str) for str in x['path_s

page_feature_rdd.cache()
page_feature_rdd.count()
```

```
10802408
```

```
%pyspark                                                                    READY

import numpy as np
from sklearn.manifold import TSNE

ndomains = 10
minpaths = 100

some_domains = page_feature_rdd\
                .filter(lambda x: len(x[1]) >= minpaths)\
                .takeSample(False, ndomains)

mat = []
for dom in some_domains:
    mat += dom[1]
mat = np.array(mat)

lookup = [(x[0], len(x[1])) for x in some_domains]
col = []
for i in range(len(lookup)):
    _, ct = lookup[i]
    col += [[i] for j in range(ct)]

proj_2d = TSNE(n_components=2).fit_transform(mat)


import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(15,10))
cax = ax.scatter(proj_2d[:,0], proj_2d[:,1], s=5.0, c=col, edgecolors='face', cmap='rainbo
cbar = fig.colorbar(cax, ticks=range(ndomains), shrink=0.7)
cbar.ax.set_yticklabels([dom[0] for dom in some_domains])  # vertically oriented colorbar
```

```
plt.show()
```




```
%pyspark
```

READY