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 0 sec. Last updated by anonymous at September 08 2017, 10:13:32 AM.

```
%pyspark
```
FINISHED

```
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)
```

Took 3 min 50 sec. Last updated by anonymous at September 08 2017, 10:17:26 AM.

```
%pyspark
```
FINISHED

```
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
```

Took 2 sec. Last updated by anonymous at September 08 2017, 10:19:02 AM.

We next define a string encoding of domains. READY
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 08 2017, 11:03:31 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
```
FINISHED

```
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
```

Took 2 sec. Last updated by anonymous at September 08 2017, 10:19:45 AM.

```
%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

```
 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 73 74 72 61 66 73 61 63 68 65 6e 2e 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
```

Took 1 sec. Last updated by anonymous at September 08 2017, 10:19:51 AM.

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

from collections import Counter

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

READY

```
                                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))
```

```
       9    饰    e9 - a5 - b0
       9        ef - bf - bd
       8    ¨          c2 - a8
       8    ½          c2 - bd
       8    Á          c3 - 81
       8    Û          c3 - 9b
       8    æ          c3 - a6
       8    õ          c3 - b5
       8    Œ          c5 - 92
       8    ˜          cb - 9c
       8    ̈          cc - 88
       8    K          ce - 9a
       8    ת          d7 - 9e
       8    ١          db - b1
       8    ٢          db - b2
       8    ٣          db - b3
       8    ३    e0 - a4 - 89
```

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)
```

```
22       23
24     1291
25  1122548
26     3063
27      750
28     3561
29     3541
2a     2206
2b    31840
2c    27391
2d  3013177
2e  1190819
2f  5123801
30  1485864
```

```
31  1430951
32  1146185
33   837275
```

```
%pyspark

records.unpersist()
```

READY

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

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)
```

FINISHED

Took 1 hrs 31 min 50 sec. Last updated by anonymous at September 08 2017, 3:15:35 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 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
```

FINISHED

```
inputURI = "s3://billsdata.net/CommonCrawl/domain_paths_from_%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])
```

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

Took 2 min 2 sec. Last updated by anonymous at September 08 2017, 3:43:51 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_from_%d_WAT_files" % nfiles
codec = "org.apache.hadoop.io.compress.GzipCodec"
big_domain_string.saveAsTextFile(outputURI, codec)
```

FINISHED

Took 31 min 41 sec. Last updated by anonymous at September 08 2017, 4:15:56 PM.

To concatenate into a single gzip file (may need to mount extra local disk space): FINISHED

```
$ aws s3 sync
s3://billsdata.net/CommonCrawl/domain_hex_strings_from_128_WAT_files/ ./tmp
$ gunzip -c ./tmp/part*.gz | cat | gzip -c > ./tmp/big_domain_string_128.gz
$ aws s3 sync ./tmp/big_domain_string_128.gz s3://billsdata.net/CommonCrawl/
$ rm -r ./tmp
```

Took 0 sec. Last updated by anonymous at September 08 2017, 4:27:55 PM.

```
%pyspark

nfiles = 128

path_inputURI = "s3://billsdata.net/CommonCrawl/domain_paths_from_%d_WAT_files/" % nfiles
hex_inputURI = "s3://billsdata.net/CommonCrawl/domain_hex_strings_from_%d_WAT_files/" % nf
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 52 sec. Last updated by anonymous at September 08 2017, 4:29:51 PM.

Compute the distribution of characters for a small sample: READY

```
%pyspark

from collections import Counter

big_path_sample = domain_paths.take(10000)

# 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_path_sample)\
        .map(lambda s: Counter(s))\
        .aggregate(Counter(),
                    lambda acc, value: acc + value,
                    lambda acc1, acc2: acc1 + acc2)

# convert to dict:
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))
```

ERROR

```
java.net.ConnectException: Connection refused (Connection refused)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:2
06)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at org.apache.thrift.transport.TSocket.open(TSocket.java:182)
        at org.apache.zeppelin.interpreter.remote.ClientFactory.create(ClientFactory.java:5
1)
        at org.apache.zeppelin.interpreter.remote.ClientFactory.create(ClientFactory.java:3
7)
        at org.apache.commons.pool2.BasePooledObjectFactory.makeObject(BasePooledObjectFact
ory.java:60)
        at org.apache.commons.pool2.impl.GenericObjectPool.create(GenericObjectPool.java:86
1)
        at org.apache.commons.pool2.impl.GenericObjectPool.borrowObject(GenericObjectPool.j
ava:435)
        at org.apache.commons.pool2.impl.GenericObjectPool.borrowObject(GenericObjectPool.j
ava:363)
        at org.apache.zeppelin.interpreter.remote.RemoteInterpreterProcess.getClient(Remote
InterpreterProcess.java:92)
        at org.apache.zeppelin.interpreter.remote.RemoteInterpreter.interpret(RemoteInterpr
eter.java:352)
```

```
        at org.apache.zeppelin.interpreter.LazyOpenInterpreter.interpret(LazyOpenInterprete
r.java:97)
        at org.apache.zeppelin.notebook.Paragraph.jobRun(Paragraph.java:406)
        at org.apache.zeppelin.scheduler.Job.run(Job.java:175)
        at org.apache.zeppelin.scheduler.RemoteScheduler$JobRunner.run(RemoteScheduler.jav
a:329)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201
(ScheduledThreadPoolExecutor.java:180)
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Schedul
edThreadPoolExecutor.java:293)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
```

Took 0 sec. Last updated by anonymous at September 08 2017, 4:31:29 PM. (outdated)

FINISHED

```
%pyspark

from collections import Counter

big_string_sample = domain_strings.take(2000)

hex_count = sc.parallelize(big_string_sample)\
        .map(lambda s: Counter(s.split()))\
        .aggregate(Counter(),
                    lambda acc, value: acc + value,
                    lambda acc1, acc2: acc1 + acc2)

# convert to dict:
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:', 174)
 -     5338
 .    25284
0a        2
0d        1
20       61
21        4
22        2
25    35056
26      242
27        2
28       33
29       32
2b      242
2c      100
2d    49338
2e    15042
2f    63197
```

Took 0 sec. Last updated by anonymous at September 08 2017, 10:38:54 AM.

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

READY

```
%pyspark
```
FINISHED

```
import re
from math import log
from collections import Counter

def hx(i):
    a = hex(x)[2:]
    if len(a)<2: a = ''.join(['0',a])
    return a
hexabet = [hx(x) for x in range(256)] + ['.','-']

def string_features_v1(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
    return [log(N), a, b, c, d, num, cap]

def string_features_hex(hexstr):
    out = dict([(x,0) for x in hexabet])
    ct = dict(Counter(hexstr.split()))
    for k in out.keys():
        if k in ct.keys():
            out[k] += ct[k]
    out = [v[1] for v in sorted(out.iteritems(), key=lambda (k,v): k)]
    out = [x/float(sum(out)) for x in out]
    return out

def string_features_v2(str):
    return string_features_hex(hexalise(str))
```

Took 0 sec. Last updated by anonymous at September 08 2017, 11:42:38 AM. (outdated)

```
%pyspark
```
FINISHED

```
def mapper(x):
    str_set = [s for s in x['path_set'] if (string_features_v1(s) is not None) and (string
    a = [string_features_v1(s) for s in str_set]
    b = [string_features_v2(s) for s in str_set]
    return (x['domain'], a, b)

page_feature_rdd = domains_rdd.map(mapper)
page_feature_rdd.cache()
print(page_feature_rdd.count())
```

```
168033
```

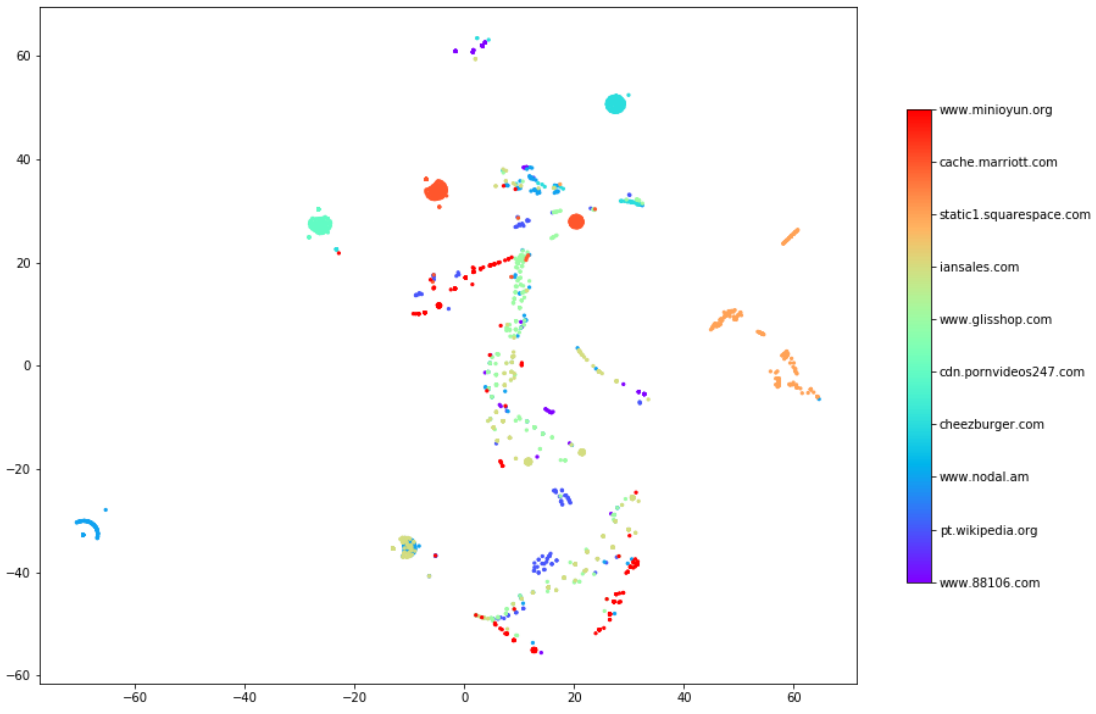
Took 16 min 15 sec. Last updated by anonymous at September 08 2017, 1:20:18 PM.

```
%pyspark
```
FINISHED

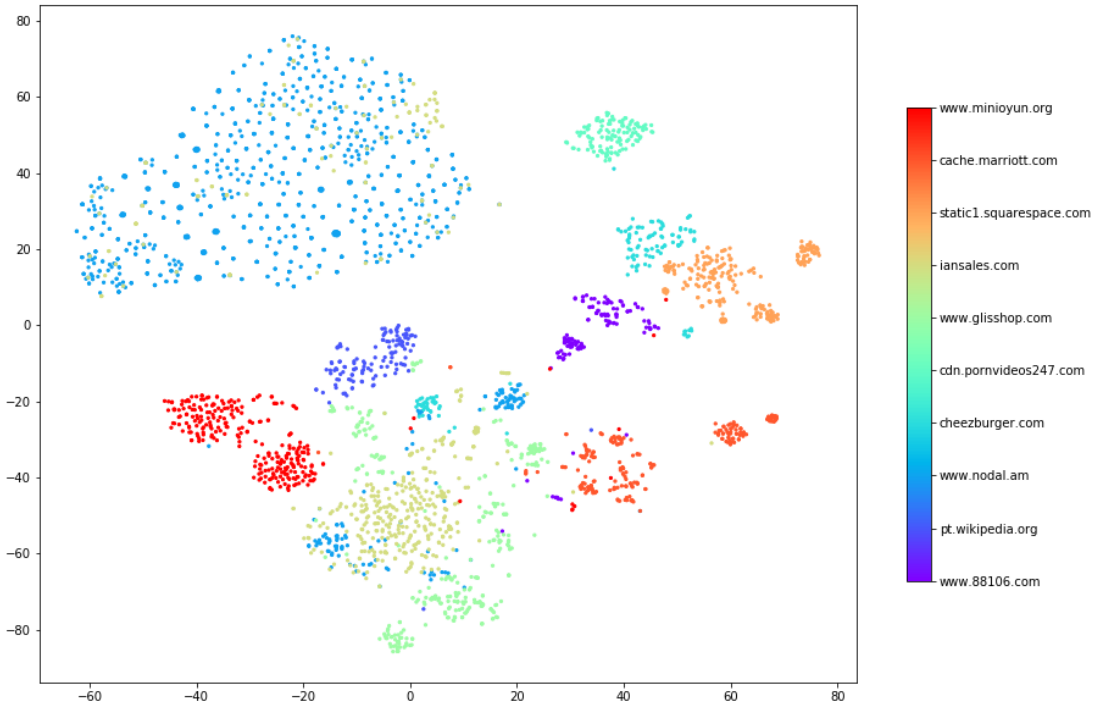
```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

ndomains = 10
minpaths = 100
```

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

mat_v1 = []
for dom in some_domains:
    mat_v1 += dom[1]
mat_v1 = np.array(mat_v1)

mat_v2 = []
for dom in some_domains:
    mat_v2 += dom[2]
mat_v2 = np.array(mat_v2)

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_v1 = TSNE(n_components=2).fit_transform(mat_v1)
proj_2d_v2 = TSNE(n_components=2).fit_transform(mat_v2)

for proj in [proj_2d_v1, proj_2d_v2]:
    fig, ax = plt.subplots(figsize=(15,10))
    cax = ax.scatter(proj[:,0], proj[:,1], s=5.0, c=col, edgecolors='face', cmap='rainbow'
    cbar = fig.colorbar(cax, ticks=range(ndomains), shrink=0.7)
    cbar.ax.set_yticklabels([dom[0] for dom in some_domains])  # vertically oriented color
    plt.show()
```

```
[<matplotlib.text.Text object at 0x7fe5b68d4690>, <matplotlib.text.Text object at 0x7fe5b69
6da90>, <matplotlib.text.Text object at 0x7fe5b68eba10>, <matplotlib.text.Text object at 0x
7fe5b68e2150>, <matplotlib.text.Text object at 0x7fe5b6a25150>, <matplotlib.text.Text objec
t at 0x7fe5b62c7c90>, <matplotlib.text.Text object at 0x7fe5b68e2390>, <matplotlib.text.Tex
t object at 0x7fe5b62d1ed0>, <matplotlib.text.Text object at 0x7fe5b68f4950>, <matplotlib.t
ext.Text object at 0x7fe5b69e4410>]
```



[, , , , , , , , ,]



Took 2 min 19 sec. Last updated by anonymous at September 08 2017, 1:43:14 PM.

```
%pyspark
```

READY