Bill 6 - simple dom...

READY

Building domain features from WAT

FINISHED

```
%pyspark

from __future__ import print_function

nfiles = 1
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:          168033
Nr page URIs:       1782572
Nr URI chars:      63676121
```

Took 14 sec. Last updated by anonymous at September 09 2017, 5:36:13 PM.

Write to S3 a single string for all domains:

FINISHED

Took 1 sec. Last updated by anonymous at September 09 2017, 5:36:26 PM.

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)
```

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

READY

```
$ 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
```

READY

URI paths and hex string RDDs

Continue with cached `domains_rdd` as above.

```
%pyspark

hex_inputURI = "s3://billsdata.net/CommonCrawl/domain_hex_strings_from_%d_WAT_files/" % nf
domain_strings = sc.textFile(hex_inputURI)
domain_strings.cache()

print(domains_rdd.count())
print(domain_strings.count())
```

READY

```
168033
168033
```

Compute the distribution of characters for a small sample:

READY

```
%pyspark

from collections import Counter

big_path_sample = domains_rdd.take(10000)

# either:
"""
char_count = sc.parallelize(big_path_sample)\
        .flatMap(lambda s: Counter(s).items())\
        .reduceByKey(lambda x,y: x+y)\
        .collect()
"""
# or:
char_count = sc.parallelize(big_path_sample)\
        .map(lambda x: ''.join(list(x['path_set'])))\
        .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))
```

READY

```
 493309    i               69
 490928    s               73
 452008    o               6f
```

```
408334    r               72
392557    n               6e
378286    l               6c
336452    m               6d
302973    0               30
302437    g               67
299488    p               70
299057    1               31
289722    c               63
271858    d               64
243261    h               68
231367    u               75
215775    2               32
179603    b               62
175818    3               33
```

```
%pyspark
```

READY

```
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:', 167)
 -     1056
 .    49954
0a       24
0d       22
20      195
21      201
25    33168
26       18
27       19
28       15
29       15
2b      317
2c      208
2d    96643
2e    36674
2f   177117
30    57788
```

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):
    """
    Normalised 2-char hex representation of 0-255
    """
    a = hex(i)[2:]
    if len(a)<2: a = ''.join(['0',a])
    return a

hexabet = [hx(x) for x in range(256)] + ['.','-']

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

def string_features_v1(str):
    """
    Coarse first version of a feature vector for a string.
    A placeholder for stronger versions.
    """
    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):
    """
    Symbol distribution of a hexalised string.
    """
    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 = [float(x)/sum(out) for x in out]
    return out

def string_features_v2(str):
    """
    Version 2: combine the hexal distribution with the previous string statistics.
```

```
    """
    N = float(len(str))
    if N==0: return None
    cap = len(re.findall(r'[A-Z]', str))/N
    num = len(re.findall(r'[0-9]', str))/N
    return string_features_hex(hexalise(str)) + [num, cap, log(N)]
```

Took 0 sec. Last updated by anonymous at September 09 2017, 5:38:01 PM.

```
%pyspark
```

READY

```
def feature_extractor(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(feature_extractor)
page_feature_rdd.cache()
print(page_feature_rdd.count())
```

```
168033
```

The plot below take a random sample of domains, and apply featrue vectors v1 and v2 to the path URIs for each domain. Dots are URIs, colours are domains:

READY

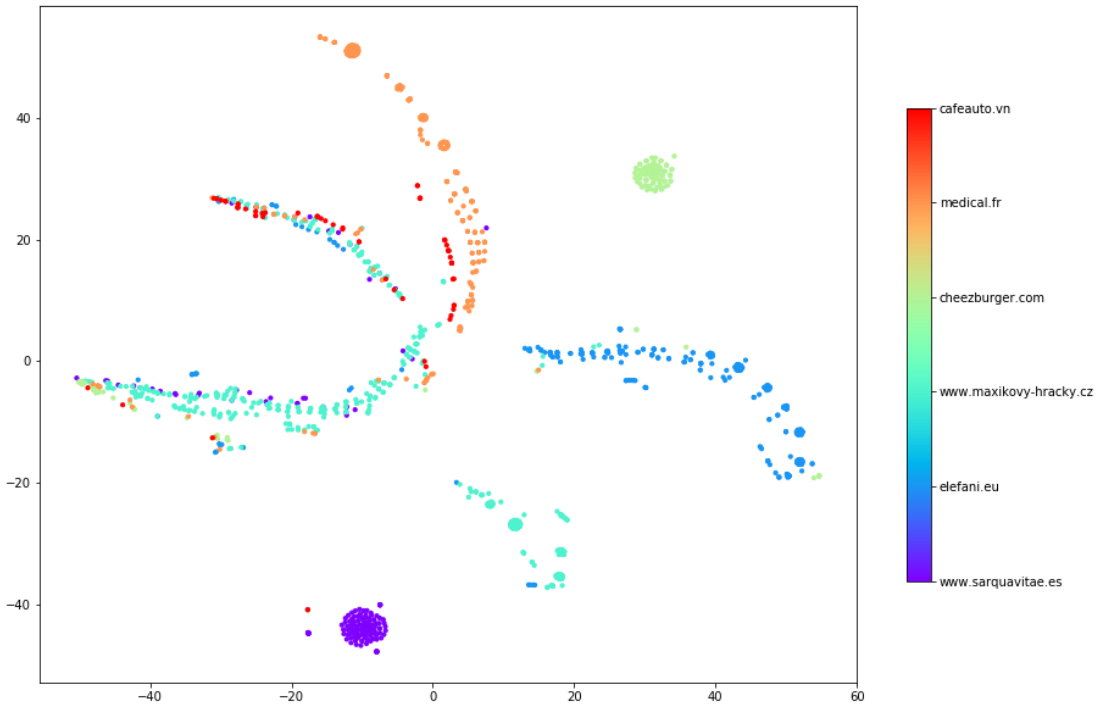
```
%pyspark
```

READY

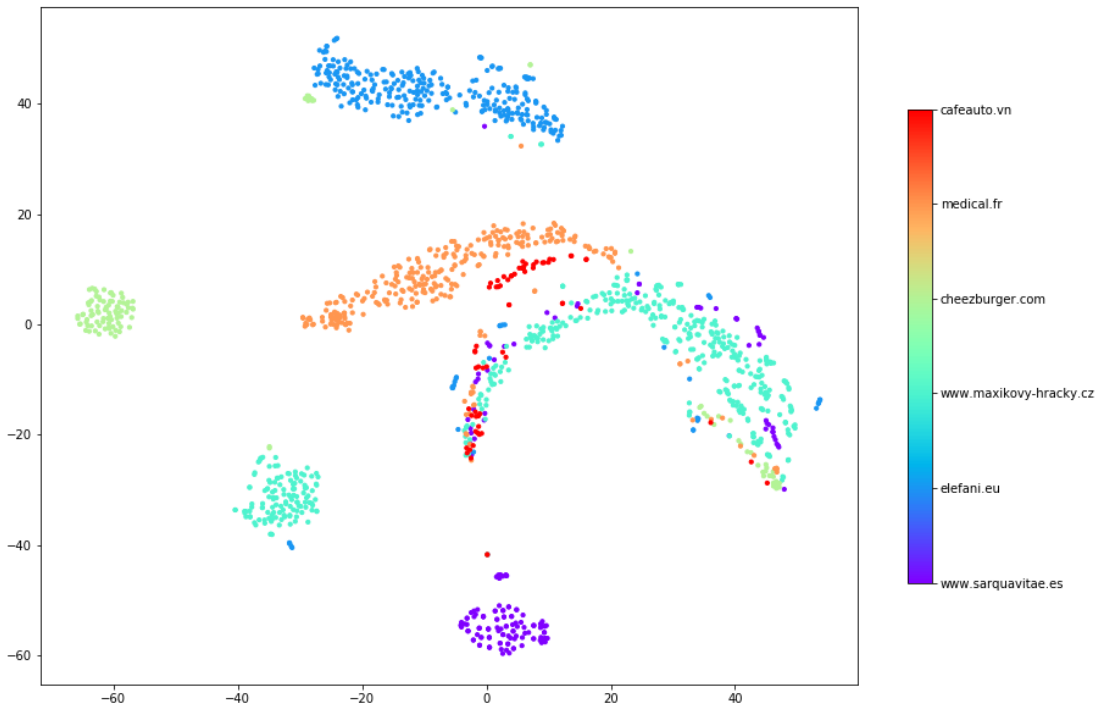
```
import numpy as np
from sklearn.manifold import TSNE as tSNE
import matplotlib.pyplot as plt

ndomains = 6
minpaths = 50

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=10.0, 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 0x7f7f7c01c310>, <matplotlib.text.Text object at 0x7f7f7c0
38f50>, <matplotlib.text.Text object at 0x7f7f7c0c24d0>, <matplotlib.text.Text object at 0x
7f7f7c0c2bd0>, <matplotlib.text.Text object at 0x7f7f7c0d0310>, <matplotlib.text.Text objec
t at 0x7f7f7c0d0a10>]
```



[, , , ,]



READY

```
%pyspark

page_feature_rdd.unpersist()
domains_rdd.unpersist()
```

```
PythonRDD[70] at RDD at PythonRDD.scala:48
```

READY

Export domain feature vectors

READY

```
%pyspark

nfiles = 128
inputURI = "s3://billsdata.net/CommonCrawl/domain_paths_from_%d_WAT_files/" % nfiles
domains_rdd = sc.textFile(inputURI).map(eval)
domains_rdd.cache()

def domain_features(domain, path_set):
    """
    Takes domain + set of paths as output by parse_urls() and
    applies extracts statistics of the signature string.
    """
    return string_features_v2(domain_string(domain, path_set))

def feature_extractor(x):
    return (x['domain'], domain_features(x['domain'], x['path_set']))

domain_feature_rdd = domains_rdd.map(feature_extractor)
```

```
%pyspark

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

READY

Timings:

READY

Cluster	nr files	nr domains	time
16 x m4.large	128	2.6M	40 min 7 sec

Let's check what we've just written:

```
%pyspark

inputURI = "s3://billsdata.net/CommonCrawl/domain_hex_feature_vectors_from_%d_WAT_files" %
features_rdd = sc.textFile(inputURI).map(eval)
print("Nr domains:", features_rdd.count())
print(features_rdd.take(1))
```

READY

```
('Nr domains:', 2626203)
[(u'www.iggl.de', [3.6375861597263857, 0.5, 0.0, 0.0, 0.02564102564102564, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.358974358974359, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.05128205128205128, 0.0, 0.0, 0.0, 0.05
128205128205128, 0.0, 0.02564102564102564, 0.02564102564102564, 0.15384615384615385, 0.2051
2820512820512, 0.0, 0.02564102564102564, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.02564102564102564, 0.0,
 0.05128205128205128, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]])
```

```
%pyspark
```

READY