201709 evaluate C...

```
%pyspark                                                                    FINISHED

# Simple script to demonstrate evaluation of CommonCrawl-derived domain vectors by usin
# classify domains according to high-level topic in the DMOZ dataset. Currently configu
# Bill's domain hex feature vectors from the 'Bill 6' notebook.
# TODO: Should we really be trying to predict domain links instead?
# PJ - 14 Sept 2017

import csv
import boto
from pyspark.sql.types import *

# Import the DMOZ domain category dataset
# (downloaded from https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910

dmoz_labels=sc.textFile('s3://billsdata.net/CommonCrawl/DMOZ/dmoz_domain_category.csv')
header = dmoz_labels.first() # extract header
dmoz_labels = dmoz_labels.filter(lambda row: row != header) # remove header row
dmoz_labels.take(1)
```

```
[u'"sdcastroverde.com","Top/World/Galego/regional/Galicia/Lugo/municipalities/Castroverd
e"']
```

```
%pyspark                                                                    FINISHED

# For now, collect all labels into a list on one node
# TODO: Could probably do this much faster using map-reduce!
dmoz_labels_list = dmoz_labels.collect()

# Take a look at one record
dmoz_labels_list[1]
```

```
u'"www.232analyzer.com","Top/Computers/Hardware/Test_Equipment/Analyzers"'
```

```
%pyspark                                                                    FINISHED

# Make a dictionary of short domains (without www.) to top-level category label, as per
# http://dmoztools.net
labels={}
prefix="www."

# TODO: Could probably do this much faster using map-reduce!
print(len(dmoz_labels_list))
for row in dmoz_labels_list[1:900000]: # Sample initially for speed (increasing to 1M c
```

```
    row = row.replace('"','').split(',')
    fulldomain = row[0]
    shortdomain = fulldomain[len(prefix):] if fulldomain.startswith(prefix) else fulldo
    label = row[1].split("/")[1].split("|")[0]
    labels[shortdomain]=label
    #print(shortdomain + " " + label)

# Take a look at the category for one domain from our dictionary
```

```
2488259
u'Computers'
```

```
%pyspark

# Summarize categories in the DMOZ data
from collections import Counter
Counter(labels.values())
```

FINISHED

```
Counter({u'World': 462130, u'Regional': 231978, u'Business': 54054, u'Society': 28807, u
'Arts': 24093, u'Shopping': 19102, u'Recreation': 16737, u'Computers': 16513, u'Sports':
12248, u'Science': 9857, u'Health': 8874, u'Reference': 7934, u'Games': 3730, u'Home': 2
549, u'News': 1383})
```

```
%pyspark

# Load Bill's domain feature vectors from s3, in the following format:
# (u'www.angelinajolin.com', [4.30406509320417, 0.02702702702702703, 0.0, 0.13513513513

nfiles=128

# Load feature vectors from WAT files (from 'Bill 6' notebook):
inputURI = "s3://billsdata.net/CommonCrawl/domain_hex_feature_vectors_from_%d_WAT_files
features_rdd = sc.textFile(inputURI).map(eval)
features_rdd.cache()
print("Nr domains:", features_rdd.count())
print(features_rdd.take(1))
```

FINISHED

```
('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, 0.05128205128205128, 0.0, 0.02564102564102564, 0.02564102564102564, 0.153846153846153
85, 0.20512820512820512, 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, 0.025641
02564102564, 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
```

```
%pyspark                                                                        FINISHED

# Convert to a python dictionary for ease-of-use initially
# TODO: Find a better Spark way to do this!
features_sample = features_rdd.sample(0, 0.15, seed=42) # TODO: Investigate memory erro
features_dict = features_sample.collectAsMap()
#features_dict['232analyzer.com']
print(len(features_dict.keys()))
features_dict.itervalues().next() # Output one vector for testing
```

```
393245
[3.912023005428146, 0.48, 0.0, 0.0, 0.0196078431372549, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.35294117647058826, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0392156862745098, 0.0, 0.0196078431372549, 0.0,
0.0784313725490196, 0.0, 0.0196078431372549, 0.0196078431372549, 0.13725490196078433, 0.
17647058823529413, 0.0, 0.0196078431372549, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0588
23529411764705, 0.058823529411764705, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
```

```
%pyspark                                                                        FINISHED

# Filter embeddings for only those vectors that have entries in the DMOZ dictionary (i.
```

```
new_vec_ids=[]
new_vec_embs=[]
ground_truth=[]

def intersect(a, b):
     return list(set(a) & set(b))

common_domains=intersect(features_dict.keys(), labels.keys())
print(len(common_domains))
print(common_domains[1])

# Iterate over all the domain IDs for which we also have a vector embedding
for domain in common_domains:

    new_vec_ids.append(domain)
    new_vec_embs.append(features_dict[domain])
    ground_truth.append(labels[domain])

# Verify lengths of each list
```

```
3057
privateerpress.com
3057 3057 3057
```

```
%pyspark

# Split into training and test sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(new_vec_embs, ground_truth, test_si
```

FINISHED

```
%pyspark

# Summarize labels in our test data
Counter(y_test)
```

FINISHED

```
Counter({u'World': 587, u'Regional': 283, u'Computers': 103, u'Arts': 87, u'Society': 86
, u'Business': 71, u'Reference': 69, u'Recreation': 49, u'Science': 41, u'Shopping': 29,
u'Sports': 28, u'Health': 25, u'Games': 25, u'News': 23, u'Home': 23})
```

```
%pyspark

# Fit KNN classifier to the training data and report results on test set
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3, metric='cosine', algorithm='brute')
neigh.fit(X_train, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_test, neigh.predict(X_test)))
```

FINISHED

```
             precision    recall  f1-score   support
       Arts       0.05      0.13      0.07        87
   Business       0.09      0.17      0.12        71
  Computers       0.11      0.17      0.13       103
      Games       0.02      0.04      0.03        25
     Health       0.00      0.00      0.00        25
       Home       0.03      0.04      0.04        23
       News       0.06      0.04      0.05        23
 Recreation       0.03      0.02      0.02        49
  Reference       0.07      0.09      0.08        69
   Regional       0.23      0.22      0.23       283
    Science       0.11      0.05      0.07        41
   Shopping       0.00      0.00      0.00        29
    Society       0.16      0.06      0.09        86
     Sports       0.00      0.00      0.00        28
      World       0.50      0.38      0.43       587
avg / total       0.27      0.22      0.24      1529
```

```
%pyspark

# Fit Random Forest classifier to the training data and report results on test set
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(max_depth=2, random_state=0)
rf.fit(X_train, y_train)
print(classification_report(y_test, rf.predict(X_test)))
```

FINISHED

```
             precision    recall  f1-score   support
       Arts       0.00      0.00      0.00        87
   Business       0.00      0.00      0.00        71
  Computers       0.00      0.00      0.00       103
      Games       0.00      0.00      0.00        25
     Health       0.00      0.00      0.00        25
       Home       0.00      0.00      0.00        23
       News       0.00      0.00      0.00        23
 Recreation       0.00      0.00      0.00        49
  Reference       0.00      0.00      0.00        69
   Regional       0.00      0.00      0.00       283
    Science       0.00      0.00      0.00        41
   Shopping       0.00      0.00      0.00        29
    Society       0.00      0.00      0.00        86
     Sports       0.00      0.00      0.00        28
      World       0.38      1.00      0.55       587
avg / total       0.15      0.38      0.21      1529
```

```
%pyspark
```

READY