201709 evaluate C...

```
%pyspark                                                                                         FINISHED

# Zeppelin notebook to demonstrate evaluation of CommonCrawl-derived domain vectors by u
# classify domains according to high-level topic in the DMOZ dataset. Currently configu
# Bill's domain hex feature vectors from the 'Bill 6' notebook, and to use only Pyspark
# All cells should complete in less than a few minutes on an m4.2xlarge cluster.
# End-to-end run-time: approx 30 mins, with nfiles=128.
# NOTE: Should we really be trying to predict domain links instead? Or predicting bad d
# PJ - 20 Sept 2017

import boto
from pyspark.sql.types import *

# Import the DMOZ domain category dataset as an RDD
# (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).map(lambda row: row.replace
dmoz_labels.take(3)
```

```
[[u'sdcastroverde.com', u'Top/World/Galego/regional/Galicia/Lugo/municipalities/Castrove
rde'], [u'www.232analyzer.com', u'Top/Computers/Hardware/Test_Equipment/Analyzers'], [u'
zschachwitz-tischtennis.de', u'Top/World/Deutsch/Sport/ball_Sports/table_tennis/Teams/Ge
rmany/Saxony']]
```

```
%pyspark                                                                                         FINISHED

# Convert our labels RDD into a Spark DataFrame with a schema - neither column can be Nu
schema=StructType([StructField("domain", StringType(), False), StructField("categories"
dmoz_labels_df=spark.createDataFrame(dmoz_labels,schema)
dmoz_labels_df.printSchema()
print(dmoz_labels_df.count())
dmoz_labels_df.show(1)
dmoz_labels_df.cache()
```

```
root
 |-- domain: string (nullable = false)
 |-- categories: string (nullable = false)
2488259
+-----------------+--------------------+
|           domain|          categories|
+-----------------+--------------------+
|sdcastroverde.com|Top/World/Galego/...|
+-----------------+--------------------+
only showing top 1 row
DataFrame[domain: string, categories: string]
```

```
%pyspark                                                                          FINISHED

# Make a dictionary of short domains (removing www. prefix) to top-level category label
prefix="www."
dmoz_labels_clean=dmoz_labels_df.rdd.map(lambda row: ((row['domain'][len(prefix):] if r
                                                       row['categories'].split("/")[1].
dmoz_labels_df.unpersist()
schema=StructType([StructField("domain", StringType(), False), StructField("category", 
dmoz_labels_clean_df=spark.createDataFrame(dmoz_labels_clean,schema)
dmoz_labels_clean_df.show(2)
dmoz_labels_clean_df.cache()
```

```
+-----------------+---------+
|           domain| category|
+-----------------+---------+
|sdcastroverde.com|    World|
|  232analyzer.com|Computers|
+-----------------+---------+
only showing top 2 rows
DataFrame[domain: string, category: string]
```

```
%pyspark                                                                          FINISHED

# Summarize categories in the DMOZ data
dmoz_labels_clean_df.groupBy('category').count().show()
```

```
|  category|  count|
+----------+-------+
|Recreation|  46095|
|     World|1273970|
|   Science|  28138|
|      Home|   6952|
| Computers|  45194|
|    Sports|  34890|
|    Health|  24218|
|   Society|  82079|
|  Shopping|  54062|
| Reference|  21663|
|     Games|  10246|
|      Arts|  66721|
|  Business| 148144|
|  Regional| 642176|
|      News|   3711|
+----------+-------+
```

```
%pyspark                                                                          FINISHED

# 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 # (takes about 5 mins for 128 files)

# Load feature vectors from WAT files (from 'Bill 6' notebook) as an RDD:
inputURI = "s3://billsdata.net/CommonCrawl/domain_hex_feature_vectors_from_%d_WAT_files
features_rdd = sc.textFile(inputURI).map(eval)
import pyspark.sql.types as typ
schema=StructType([StructField("domain", StringType(), False), StructField("vector", Ar
features_df=spark.createDataFrame(features_rdd,schema)
features_df.cache()
print("Nr domains:", features_df.count())
print(features_df.show(1))
features_df.printSchema()
```

```
('Nr domains:', 2626203)
+-----------+--------------------+
|     domain|              vector|
+-----------+--------------------+
|www.iggl.de|[3.63758615972638...|
+-----------+--------------------+
only showing top 1 row
None
root
 |-- domain: string (nullable = false)
 |-- vector: array (nullable = true)
 |    |-- element: double (containsNull = false)
```

```
%pyspark
```

FINISHED

```
# Spark.ML classifiers require VectorUDF type, rather than Array, so we need to convert
from pyspark.ml.linalg import Vectors, VectorUDT
vectorize=udf(lambda vs: Vectors.dense(vs), VectorUDT())
features_df = features_df.withColumn("vec", vectorize(features_df['vector'])).drop('vec
print(features_df.show(1))
features_df.printSchema()
```

```
+-----------+--------------------+
|     domain|                 vec|
+-----------+--------------------+
|www.iggl.de|[3.63758615972638...|
+-----------+--------------------+
only showing top 1 row
None
root
 |-- domain: string (nullable = false)
 |-- vec: vector (nullable = true)
```

```
%pyspark
```

FINISHED

```
# Filter embeddings for only those vectors that have entries in the DMOZ dictionary (i.
#common_domains_df= features_df.join(dmoz_labels_clean_df, features_df.domain == dmoz_l
common_domains_df=features_df.join(dmoz_labels_clean_df, ["domain"]) # doesn't create e
```

```
common_domains_df.cache()
features_df.unpersist()
dmoz_labels_clean_df.unpersist()
print("Number of labelled domains = " + str(common_domains_df.count()))
common_domains_df.show(3)
```

```
Number of labelled domains = 57479
+---------+--------------------+--------+
|   domain|                 vec|category|
+---------+--------------------+--------+
|   1by.by|[4.74493212836325...|   World|
|   360.ch|[5.42934562895444...|   World|
|631la.com|[3.46573590279972...|   World|
+---------+--------------------+--------+
only showing top 3 rows
root
 |-- domain: string (nullable = false)
 |-- vec: vector (nullable = true)
 |-- category: string (nullable = false)
```

```
%pyspark                                                                          FINISHED

# Create numeric indexes for our classes
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
labelIndexer = StringIndexer(inputCol="category", outputCol="indexedCategory").fit(comm

# Split into training and test sets using spark.ML API
domains_train, domains_test = common_domains_df.randomSplit([0.7,0.3],seed=42)
domains_test.groupBy('category').count().show()
```

```
|  category|count|
+----------+-----+
|Recreation|  419|
|     World| 6469|
|   Science|  564|
|      Home|  211|
| Computers| 1168|
|    Sports|  400|
|    Health|  233|
|   Society| 1016|
|  Shopping|  373|
| Reference|  857|
|     Games|  209|
|      Arts|  928|
|  Business|  795|
|  Regional| 3460|
|      News|  180|
+----------+-----+
```

```
%pyspark                                                                          FINISHED
```

```
# Create a pipeline and fit a RandomForest Classifier using spark.ml
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier

# Build our RF classifier
rf = RandomForestClassifier(labelCol="indexedCategory", featuresCol="vec", numTrees=10)

# Convert indexed labels back to original labels
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedCategory", lal

# Define and run the full pipeline to train the model and make predictions
pipeline = Pipeline(stages=[labelIndexer, rf, labelConverter])
model=pipeline.fit(domains_train)
predictions=model.transform(domains_test)
print(predictions.take(1))
predictions.select("predictedCategory", "category", "vec").show(5)

# FYI, Equivalent code in sklearn
#from sklearn.ensemble import RandomForestClassifier
#rf = RandomForestClassifier(max_depth=2, random_state=0)
#rf.fit(X_train, y_train)
```

```
[Row(domain=u'360.ch', vec=DenseVector([5.4293, 0.5307, 0.0, 0.0, 0.0044, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3362, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0131, 0.0, 0.0262, 0.0262, 0.0873, 0.0655, 0
.0131, 0.0437, 0.1659, 0.0742, 0.0131, 0.0131, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0044, 0.0, 0.0087, 0.
0306, 0.0218, 0.0524, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]), catego
ry=u'World', indexedCategory=0.0, rawPrediction=DenseVector([3.9293, 1.8593, 0.6566, 0.6
234, 0.5674, 0.4469, 0.4028, 0.3395, 0.2174, 0.236, 0.2129, 0.1511, 0.1064, 0.1276, 0.12
```

```
%pyspark                                                                        FINISHED

# Select (prediction, true label) and compute test error
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator1 = MulticlassClassificationEvaluator(labelCol="indexedCategory", predictionCo
evaluator2 = MulticlassClassificationEvaluator(labelCol="indexedCategory", predictionCo

accuracy = evaluator1.evaluate(predictions)
f1=evaluator2.evaluate(predictions)

print("Accuracy=%g, F1=%g" % (accuracy, f1))
```

```
Accuracy=0.37432, F1=0.203905
```

```
%pyspark
```

FINISHED