Paul 3 - evaluate C…

```
%pyspark                                                                    FINISHED

# Zeppelin notebook to evaluate CC domain feature vectors against the DNS-BH Malware
# Domain Blocklist from this site: http://mirror1.malwaredomains.com
# Specifically, the 'justdomains' file, which currently contains 31k 'bad' domains.
# We train using some of these domains, and try to predict the rest from amongst all
# the domains for which we have feature vectors.
# PJ - 25 Sept 2017

import boto
from pyspark.sql.types import *

# Import the DNS-BH domain list as a DataFrame
bh_schema=StructType([StructField("domain", StringType(), False)])
dns_bh=spark.read.csv('s3://billsdata.net/CommonCrawl/DNS-BH/justdomains.dms', header=F
dns_bh.show(3)
print("Bad domains in DNS-BH: " + str(dns_bh.count()))
dns_bh.cache()
```

```
+--------------------+
|              domain|
+--------------------+
|amazon.co.uk.secu...|
|autosegurancabras...|
|christianmensfell...|
+--------------------+
only showing top 3 rows
Bad domains in DNS-BH: 31877
DataFrame[domain: string]
```

```
%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
from pyspark.sql.functions import UserDefinedFunction
vectorize=UserDefinedFunction(lambda vs: Vectors.dense(vs), VectorUDT())
features_df = features_df.withColumn("vec", vectorize(features_df['vector'])).drop('vec
features_df.show(1)
features_df.printSchema()
```

```
+-----------+--------------------+
|     domain|                 vec|
+-----------+--------------------+
|www.iggl.de|[3.63758615972638...|
+-----------+--------------------+
only showing top 1 row
root
 |-- domain: string (nullable = false)
 |-- vec: vector (nullable = true)
```

```
%pyspark
```

FINISHED

```
# Remove www. prefix from both DNS_BH and Bill's vectors
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import StringType

name='domain'
prefix="www."
udf = UserDefinedFunction(lambda x: (x[len(prefix):] if (x.startswith(prefix) if x else

dns_bh2 = dns_bh.select(*[udf(column).alias(name) if column == name else column for col
dns_bh2.show(3)

features_df2 = features_df.select(*[udf(column).alias(name) if column == name else colu
features_df2.show(3)
```

```
+--------------------+
|              domain|
+--------------------+
|amazon.co.uk.secu...|
|autosegurancabras...|
|christianmensfell...|
+--------------------+
only showing top 3 rows
+---------------+--------------------+
|          domain|                 vec|
+---------------+--------------------+
|         iggl.de|[3.63758615972638...|
|     bmskirov.ru|[3.91202300542814...|
|education.nh.gov|[4.00733318523247...|
+---------------+--------------------+
only showing top 3 rows
```

```
%pyspark                                                                    FINISHED

# Filter feature vectors for only those vectors that have entries in the DNS-BH list di
common_domains_df=features_df2.join(dns_bh2, ["domain"]) # doesn't create extra column
common_domains_df.cache()
features_df.unpersist()
dns_bh.unpersist()
print("Number of labelled domains = " + str(common_domains_df.count()))
common_domains_df.show(3)
common_domains_df.printSchema()

# We appear to only have 149 bad domains in the 128 WAT files currently processed by Bi
# TODO: Get more/better data, both for CC feature vectors, and known bad domains!
```

```
Number of labelled domains = 149
+---------------+--------------------+
|         domain|                 vec|
+---------------+--------------------+
|    tsjyoti.com|[4.57471097850338...|
|jur-science.com|[6.52356230614951...|
|      simbio.ru|[5.05624580534830...|
+---------------+--------------------+
only showing top 3 rows
root
 |-- domain: string (nullable = true)
 |-- vec: vector (nullable = true)
```

```
%pyspark                                                                    FINISHED

# Add a category column to our dataframe, and label all these as bad
from pyspark.sql.functions import lit
common_domains_df=common_domains_df.withColumn("category", lit("bad"))
common_domains_df.show(3)
#common_domains_df.printSchema()
```

```
# Now create a 'good' class made up of a random sample of other domains in Bill's data
number_of_samples=common_domains_df.count()
fraction=float(number_of_samples)/features_df2.count()
print("--> Finding " + str(number_of_samples) + " samples of good domains, fraction of
features_df2_sample=features_df2.sample(False, fraction, 42) # create roughly balanced
features_df2_sample=features_df2_sample.withColumn("category", lit("good"))
features_df2_sample.show(3)
#features_df2_sample.printSchema()

# Concatenate the two dataframes together using union, and summarize
union_df=common_domains_df.union(features_df2_sample)
union_df.groupBy('category').count().show()
```

```
|      simbio.ru|[5.0562458053483E0...|     bad|
+---------------+---------------------+--------+
only showing top 3 rows
--> Finding 149 samples of good domains, fraction of 5.67359035078e-05
+--------------------+--------------------+--------+
|              domain|                 vec|category|
+--------------------+--------------------+--------+
|  pindao.blogbus.com|[4.07753744390572...|    good|
|qhkPtNVo.www.00vg...|[4.29045944114839...|    good|
|    bursa.shalala.ru|[4.00733318523247...|    good|
+--------------------+--------------------+--------+
only showing top 3 rows
+--------+-----+
|category|count|
+--------+-----+
|     bad|  149|
|    good|  179|
+--------+-----+
```

```
%pyspark

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

# Split into training and test sets using spark.ML API
domains_train, domains_test = union_df.randomSplit([0.7,0.3],seed=42)

# Let's check the breakdown of categories in our test data
domains_test.groupBy('category').count().show()
```

FINISHED

```
+--------+-----+
|category|count|
+--------+-----+
|     bad|   49|
|    good|   53|
+--------+-----+
```

```
%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
```

```
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0]), category=u'bad', indexedCategory=1.0, rawPrediction=DenseVector([5.1427, 4.8573]),
probability=DenseVector([0.5143, 0.4857]), prediction=0.0, predictedCategory=u'good')]
+-----------------+--------+--------------------+
|predictedCategory|category|                 vec|
+-----------------+--------+--------------------+
|             good|     bad|[4.26267987704131...|
|              bad|     bad|[6.52356230614951...|
|             good|     bad|[3.89182029811062...|
|              bad|     bad|[9.70899219481046...|
|             good|     bad|[7.86403565907245...|
```

```
%pyspark                                                                    FINISHED

# Select (prediction, true label) and compute overall test accuracy (not that this is m
# Binary evaluator only seems to support AUC metrics, so use Multiclass instead
#from pyspark.ml.evaluation import BinaryClassificationEvaluator
#evaluator1 = BinaryClassificationEvaluator(rawPredictionCol="prediction", labelCol="in
#evaluator2 = BinaryClassificationEvaluator(rawPredictionCol="prediction", labelCol="in
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=0.519608, F1=0.488773
```

```
%pyspark
```

FINISHED

```
# Compute the precision, recall and f1-score with respect to the bad class manually
tp=predictions[(predictions['predictedCategory']=='bad') & (predictions['category']=='b
fp=predictions[(predictions['predictedCategory']=='bad') & (predictions['category']=='g
fn=predictions[(predictions['predictedCategory']=='good') & (predictions['category']=='
print("tp="+str(tp)+",fp="+str(fp)+",fn="+str(fn))
precision=float(tp)/(tp+fp)
recall=float(tp)/(tp+fn)
f1=(2*precision*recall)/(precision+recall)
print("precision="+str(precision)+", recall="+str(recall)+", f1="+str(f1))

# So we're not doing very well yet, but wouldn't expect to with the limited feature set
```

```
tp=13,fp=13,fn=36
precision=0.5, recall=0.265306122449, f1=0.346666666667
```

```
%pyspark
```

FINISHED