Tom 2 Wiki Topic ...

```
%pyspark                                                                FINISHED

# PySpark CommonCrawl Topic Modelling
# Tom V / Paul J - 14/2/2018

# SET THE spark.driver.maxResultSize PROPERTY TO 16g

import boto
from boto.s3.key import Key
from gzipstream import GzipStreamFile
from pyspark.sql.types import *
import warc
import ujson as json
from urlparse import urlparse
from langdetect import detect_langs
import pycld2 as cld2

# Latest blog/documentation: http://commoncrawl.org/2018/01/january-2018-crawl-archive-
wetlist = sc.textFile("s3://commoncrawl/crawl-data/CC-MAIN-2018-05/wet.paths.gz") # Jan

wetlist.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 detect(x):
    try:
        return detect_langs(x)[0].lang # Maybe we can get away with looking at less cha
    except Exception as e:
        return None

def detect2(x):
    try:
        isReliable, textBytesFound, details = cld2.detect(x)
        return details[0][1]
    except Exception as e:
        print(e)
        return None

def process_wet(id_, iterator):
    for uri in iterator:
        file = unpack(uri)
        for record in file: # Approx 53k web pages per WET file
            try:
                #url = record.rec_headers.get_header('WARC-Target-URI')
                #yield record, record.content_stream().read().decode('utf-8')
                url = record.url

                # TODO: Limit number of bytes read per record e.g. read(200000)

                domain = None if not url else urlparse(url).netloc
                text = record.payload.read().decode('utf-8') #.limit(100) # TODO: Limit
                lang = detect2(text[:300]) # Use PyCLD2, not langdetect, which was kill
                yield domain, url, text, lang
            except Exception as e:
                yield e

def process_wet_simple(id_, iterator):
    count=0
    for uri in iterator:
        file = unpack(uri)
        for record in file:
            try:
                count=count+1
                # TODO: Output total size of pages, rather than number of pages
                # Histogram.
            except Exception as e:
                pass
        #print(count)
```

```
%pyspark
detect2("this is a test")
```

READY

```
%pyspark

# PARAMETER - number of input files
nfiles = 8192 # Total 80000

# PARAMETER - slices / partitions of input
files = sc.parallelize(wetlist.take(nfiles)) #, numSlices=nfiles/32) # TODO: Try numSli

# Should parallelize
print(files.getNumPartitions())
rdd=files.mapPartitionsWithIndex(process_wet)

print(str(rdd))
docs = rdd.toDF(["host", "url", "text","lang"]) #  "lang"
#docs.cache()
#docs.count() # Total docs in all languages
```

READY

```
%pyspark

# Filter for English only
docs_en = docs.filter(docs.lang == 'en')
```

READY

Load saved vectors from Wikipedia model (created by python Wikipedia Text Processing.ipynb)

READY

```
%pyspark
from pyspark.ml import Pipeline,PipelineModel
from pyspark.ml.feature import RegexTokenizer, CountVectorizer, StopWordsRemover
from pyspark.ml.clustering import LocalLDAModel

textModel = PipelineModel.load('s3://billsdata.net/CommonCrawl/wikipedia/text_model')
ldaModel = LocalLDAModel.load('s3://billsdata.net/CommonCrawl/wikipedia/lda_model')
```

READY

```
%pyspark

# Test the models - for debugging only
import numpy as np
import pandas as pd

X=ldaModel.topicsMatrix().toArray()
vocab = np.array(textModel.stages[2].vocabulary)

topicLabels = [' '.join(vocab[np.argsort(X[:,i])[::-1][:5]]) for i in range(100)]

def score_topics(text):
    df = sqlContext.createDataFrame(pd.DataFrame({'text':[text]}))
    vec = textModel.transform(df)
    scores = ldaModel.transform(vec).select('topicDistribution').collect()[0].topicDistr
    return pd.Series(dict(zip(topicLabels, scores)))

# Try it on an arbitary sentence
print(score_topics("This is the latest news about North Korea and their involvement in
```

READY

```
%pyspark

# Now score pages from our WET files
docs_en.show(5)
vec = textModel.transform(docs_en)
vec.show(5)
```

READY

```
%pyspark

# Create topic distribution vectors and tidy upp
scores = ldaModel.transform(vec)
scores2 = scores.drop("url").drop("text").drop("lang").drop("words").drop("filtered").d
scores2.show(5)
```

READY

```
%pyspark
```

```
# Save these vectors to disc, so we can just load them later
scores2.write.parquet('s3://billsdata.net/CommonCrawl/topic_model_%d_files/cc_page_wiki_
```

Load saved scores from nfiles of WET files

READY

```
%pyspark

# Restart here
nfiles=8192
scores2 = spark.read.parquet('s3://billsdata.net/CommonCrawl/topic_model_%d_files/cc_pag
scores2.show(5)
```

```
%pyspark
```

READY

```
# Aggregate page-scores per Host for now (will be same process for aggregating host-scor
scores3=scores2.rdd.map(lambda x: (x['host'], (1,x['topicDistribution']))).reduceByKey(l

# Next, divide by the total to create averaged vectors, and convert back to a dataframe
scores4=scores3.map(lambda x: (x[0], (x[1][1]/x[1][0]))).toDF(["host", "averageTopicDist
scores4.show(5)
{s.name: type(s.dataType) for s in scores4.schema}
print(scores2.count())
print(scores4.count())
```

```
%pyspark
```

READY

```
# Just playing - code to help understand the different libraries and vector types!
import pyspark.mllib.linalg as mllib
import pyspark.ml.linalg as ml
df = sc.parallelize([
    (mllib.DenseVector([1, ]), ml.DenseVector([1, ])),
    (mllib.SparseVector(1, [0, ], [1, ]), ml.SparseVector(1, [0, ], [1, ]))
]).toDF(["mllib_v", "ml_v"])
df.show()
{s.name: type(s.dataType) for s in df.schema}
```

```
%pyspark
```

FINISHED

```
# Enrich each row with the corresponding PLD (using code from Paul J)
#saveURI="s3://billsdata.net/CommonCrawl/hyperlinkgraph/cc-main-2017-aug-sep-oct/domain
saveURI="s3://billsdata.net/CommonCrawl/hyperlinkgraph/cc-main-2017-18-nov-dec-jan/domai
pld_df=spark.read.load(saveURI)
pld_df.show(3)
pld_df.cache()
pld_df.count() # 70M in latest web graph
```

```
+---+-------+--------+
| ID|    PLD|NumHosts|
+---+-------+--------+
|  0|  aaa.a|       1|
|  1| aaa.aa|       1|
|  2|aaa.aaa|       4|
+---+-------+--------+
only showing top 3 rows
70367284
```

```
%pyspark                                                                              READY

# Generate a Bloom Filter of PLDs from the vertex file
from pybloom import BloomFilter

# Attempt to distribute our Bloom Filter so we can build it in parallel, and save it to
def build_partial_bloom(capacity, error_rate):
    def _build_partial_bloom(record):
        bloom_filter = BloomFilter(capacity=capacity, error_rate=error_rate)
        for _,PLD,_ in record: # Just take the PLD field (not the ID or NumHosts)
            bloom_filter.add(PLD)
        yield (None, bloom_filter) # returns a Generator (i.e. a kind of Iterator that
    return _build_partial_bloom

def merge_bloom(bloom_in1, bloom_in2):
    return bloom_in1.union(bloom_in2) # The reduce function simply becomes a Union

# Our generator function for partial Bloom Filters
generate_bloom = build_partial_bloom(capacity=94000000, error_rate=0.005) # Only 70M in

# Construct the distributed BloomFilter using the PLD dataframe
bloom_filter_rdd = pld_df.rdd.mapPartitions(generate_bloom).reduceByKey(merge_bloom)

# Save/Load the Bloom Filter RDD - not working
#saveURI="s3://billsdata.net/CommonCrawl/hyperlinkgraph/cc-main-2017-18-nov-dec-jan/dom
#bloom_filter_rdd.saveAsTextFile(saveURI)
#bloom_filter_rdd=spark.read.load(saveURI)

# Collect and broadcast to cluster nodes
bloom_filter=bloom_filter_rdd.collect()
print(bloom_filter)
pld_bf_distrib = sc.broadcast(bloom_filter[0][1])

# Test contents
import sys
print(sys.getsizeof(pld_bf_distrib.value))
print(len(pld_bf_distrib.value)) # Should match number of items entered
print("aaa.aaa" in pld_bf_distrib.value) # Should be true
print("aaa.aaa.bla" in pld_bf_distrib.value) # Should be false
```

```
%pyspark                                                                              READY
```

```
from pyspark.sql.functions import udf

# Returns a Boolean to say whether PLD is a hostname in itself
def is_a_pld(hostname):
    #if hostname in pld_lookup_table:
    #if pld_lookup_table.filter(lambda a: a == hostname).count()>0:
    if hostname in pld_bf_distrib.value:
        return True
    else:
        return False

# Define a function to do the hostname->pld conversion, if the pld exists in our dictio
def convert_hostname(hostname):
    # Return hostname as-is, if this is already a PLD
    #if hostname in pld_lookup_table:
    #if pld_lookup_table.filter(lambda a: a == hostname).count()>0:
    if hostname in pld_bf_distrib.value:
        return hostname
    # Otherwise we're going to have to split it up and test the parts
    try:
        parts=hostname.split('.')
        if (len(parts)>4 and is_a_pld('.'.join(parts[0:4]))):
            return '.'.join(parts[0:4])
        if (len(parts)>3 and is_a_pld('.'.join(parts[0:3]))):
            return '.'.join(parts[0:3])
        if (len(parts)>2 and is_a_pld('.'.join(parts[0:2]))):
            return '.'.join(parts[0:2])
        if (len(parts)>1 and is_a_pld('.'.join(parts[0:1]))):
            return '.'.join(parts[0:1])
        return "ERROR" # Couldn't find a corresponding PLD - this should never happen!
    except:
        return "ERROR"

udf_convert_hostname = udf(convert_hostname, StringType())

# Test
print(convert_hostname("aaa.aaa"))
print(is_a_pld("aaa.aaa")) # Should be true
```

```
%pyspark
```
READY

```
# Function to reverse hostnames
from pyspark.sql.functions import udf
def reverse_domain(domain):
    try:
        ret =  '.'.join(reversed(domain.split('.')))
        return ret
    except:
        return "NODOMAIN"
print(reverse_domain("com.facebook"))
udf_reverse_domain = udf(reverse_domain, StringType())

# Convert hosts in Topic DF to PLDs using convert_hostname function from Paul 5.
scores5=scores4.withColumn("pld",udf_reverse_domain(udf_convert_hostname(udf_reverse_do
scores5.show(10)
```

```
%pyspark
```
READY

```
# Now we can aggregate page-scores per PLD, using a map-reduce similar to the host aggr
scores6=scores5.rdd.map(lambda x: (x['pld'], (1,x['averageTopicDistribution']))).reduce

# Next, divide by the total to create averaged vectors, and convert back to a dataframe
scores7=scores6.map(lambda x: (x[0], (x[1][1]/x[1][0]))).toDF(["pld_rev", "averageTopic
scores7.show(5)
{s.name: type(s.dataType) for s in scores7.schema}
print(scores5.count())
print(scores7.count())
```

```
%pyspark
```
READY

```
# Save pld topic distributions in parquet format for Tom to play with (and to figure ou
# TODO: Maybe a numpy argmax to get the index of the 'top' topic for each PLD with a sc
scores7.write.parquet('s3://billsdata.net/CommonCrawl/topic_model_%d_files/cc_pld_wiki_
```

Load saved PLD vectors from nfiles of WET files

FINISHED

```
%pyspark

# Restart here
nfiles=8192
scores7 = spark.read.parquet('s3://billsdata.net/CommonCrawl/topic_model_%d_files/cc_pl
scores7.show(5)
```

```
+--------------------+------------------------+
|             pld_rev|averageTopicDistribution|
+--------------------+------------------------+
|          scrtec.org|    [0.00101137139816...|
|corporatefitnessc...|    [1.16246367421139...|
|        1centweb.com|    [1.94489585421988...|
|   mygaragestory.net|    [2.04009774069444...|
|     gitedeville.com|    [1.18827049530001...|
+--------------------+------------------------+
only showing top 5 rows
```

```
%pyspark
```
FINISHED

```
# Next run t-SNE embedding on the vectors to reduce from indlen dimensions to 2
import numpy as np
from sklearn.manifold import TSNE
scores8=scores7.limit(10000)
print(scores8.count())
X=np.array(scores8.select('averageTopicDistribution').collect())
print(X)
new_X = np.array([i[0] for i in X])
print(new_X)
```

```
X_embedded = TSNE(n_components=2).fit_transform(new_X)
X_embedded.shape
```

```
 [[  2.88131243e-05   2.75475952e-05   3.04710806e-05 ...,   2.81830024e-05
     2.68391742e-05   2.78737638e-05]]
 [[  7.76791115e-04   7.42672922e-04   8.21489002e-04 ...,   7.59803265e-04
     7.23574156e-04   7.51466306e-04]]]
[[  1.01137140e-03   9.66949978e-04   1.06956744e-03 ...,   9.89253450e-04
    9.42083646e-04   9.78398843e-04]
 [  1.16246367e-04   1.11140598e-04   1.22935382e-04 ...,   1.13704145e-04
    1.08282478e-04   1.12456523e-04]
 [  1.94489585e-04   1.85947220e-04   2.05680848e-04 ...,   1.90236241e-04
    1.81165354e-04   1.88148870e-04]
 ...,
 [  1.80705922e-04   1.72768963e-04   1.91104049e-04 ...,   1.76754016e-04
    1.68325992e-04   1.74814579e-04]
 [  2.88131243e-05   2.75475952e-05   3.04710806e-05 ...,   2.81830024e-05
    2.68391742e-05   2.78737638e-05]
 [  7.76791115e-04   7.42672922e-04   8.21489002e-04 ...,   7.59803265e-04
    7.23574156e-04   7.51466306e-04]]
(10000, 2)
```

```
%pyspark                                                                    FINISHED

# Let's try the same thing in UMAP (BEWARE: install umap-learn, not umap!)
import umap
U_embedded = umap.UMAP().fit_transform(new_X)
U_embedded.shape
```

```
(10000, 2)
```

```
%pyspark                                                                    FINISHED

# Save as CSV for loading by standalone Bokeh demo (tidied up from 2_Create_Summaries_F

# Add tSNE and umap coordinates as new columns
tsne_col = sc.parallelize(X_embedded.tolist(), 1)
umap_col = sc.parallelize(U_embedded.tolist(), 1)
print(tsne_col.count(),umap_col.count(),scores8.count())
def process_tsne(pair):
   return dict(pair[0].asDict().items() + [("tSNE", pair[1])])
def process_umap(pair):
   return dict(pair[0].asDict().items() + [("umap", pair[1])])
print(scores8.count())
with_tsne_rdd = (scores8.rdd.coalesce(1).zip(tsne_col).map(process_tsne)) # Add new col
scores_tsne = sqlContext.createDataFrame(with_tsne_rdd) # Rebuild data frame
print(scores_tsne.count())
with_umap_rdd = (scores_tsne.rdd.coalesce(1).zip(umap_col).map(process_umap)) # Add new
scores_tsne_umap = sqlContext.createDataFrame(with_umap_rdd) # Rebuild data frame

# convert tSNE & umap to strings (csv export wouldn't allow them as arrays)
from pyspark.sql.functions import udf
```

```
def str_func(embedArray):
    outArray = []
    for e in embedArray:
        outArray.append(str(e))
    return ','.join(outArray)
str_udf = udf(str_func, StringType())
scores8=scores_tsne_umap.withColumn("tSNEString", str_udf(scores_tsne_umap.tSNE)).withC
```

```
(10000, 10000, 10000)
10000
10000
+----------------------+--------------------+--------------------+--------------------
+--------------------+--------------------+
|averageTopicDistribution|             pld_rev|                tSNE|                umap
|          tSNEString|          umapString|
+----------------------+--------------------+--------------------+--------------------
+--------------------+--------------------+
|    [3.15059422331988...|servprometrocrest...|[-4.4032044410705...|[-5.6035976260947...
|-4.40320444107,-7...|-5.60359762609,9....|
|    [5.27446243610823...|waynestatecollege...|[57.6027946472168...|[-0.9963978965742...
|57.6027946472,-7....|-0.996397896574,-...|
|    [5.18636606421855...|      isrbrevard.com|[2.58689379692077...|[-1.5378591424965...
|2.58689379692,-40...|-1.5378591425,4.8...|
|    [4.41861945505762...|dirttechlandscapi...|[-42.294990539550...|[-3.8410027676229...
|-42.2949905396,-9...|-3.84100276762,2....|
|    [0.00148971332672...|septicpreservatio...|[-21.944774627685...|[-0.3517232434838
```

```
%pyspark
```
FINISHED

```
# Take reversed PLD and switch back to the right way round
from pyspark.sql.functions import udf, col, when, lit, concat_ws, split
def reverse_domain(domain):
    return '.'.join(reversed(domain.split('.')))
print(reverse_domain("com.facebook.abc"))

udf_reverse_domain = udf(reverse_domain, StringType())
scores9=scores8.withColumn("actual_pld_rev",udf_reverse_domain("pld_rev"))
scores10=scores9.join(pld_df, pld_df.PLD==scores9.actual_pld_rev).withColumnRenamed("pl
#scores9=scores8.join(pld_df,"PLD").withColumnRenamed("PLD","PLD_rev").withColumn("payL
scores10.show(3)
```

```
abc.facebook.com
+--------------------+--------+--------------------+--------------------+
|      payLevelDomain|NumHosts|          tSNEString|          umapString|
+--------------------+--------+--------------------+--------------------+
|            cripe.ca|       1|-15.979842186,-36...|-1.1282559629,3.2...|
|    anythehoorah.com|       2|2.81252336502,26....|0.261848978601,-4...|
|arizonaduderanche...|       1|-2.5158662796,-79...|-5.56482661887,9....|
+--------------------+--------+--------------------+--------------------+
only showing top 3 rows
```

```
%pyspark
```
FINISHED

```
# Export as CSV
topicEmbeddingsURI="s3://billsdata.net/CommonCrawl/domain_embeddings_topic_10000_2017-1
codec="org.apache.hadoop.io.compress.GzipCodec"
scores10.coalesce(1).write.format('com.databricks.spark.csv').options(header='true', co
```

```
%pyspark
```

READY

```
# Plot the t-SNE embedding using matplotlib in Zeppelin notebook
import matplotlib
import StringIO

# Turn off interactive mode - we really want this but it leads to dependency errors on
matplotlib.use('agg',warn=False, force=True) # Removes Tkinter error (no interactive pl
from matplotlib import pyplot as plt
print "Switched to:",matplotlib.get_backend()

fig, ax = plt.subplots(nrows=1, ncols=2) #, figsize=(40,16))
#print(X_embedded)

#colors=np.array(pld_codes_with_vec_sc.select('StatusColour').collect())
#colors2 = np.array([i[0] for i in colors])
#groups=np.array(pld_codes_with_vec_sc.select('StatusClass').collect())
#groups2 = np.array([i[0] for i in groups])
#zip_list = np.array(zip(pld_codes_with_vec_sc.collect(),X_embedded))
#x_emb = zip_list[:,1]
#x=[row[0] for row in x_emb]
#y=[row[1] for row in x_emb]

#ax.scatter(X_embedded[:,0], X_embedded[:,1], s=1.0, c=colors2, edgecolors='face', cmap
#ax.scatter(x, y, s=1.0, c=colors2, edgecolors='face', cmap='rainbow')
ax[0].set_title("t-SNE embedding of PLD topic vectors")
ax[0].scatter(X_embedded[:,0], X_embedded[:,1], s=1.0)
ax[1].set_title("UMAP embedding of PLD topic vectors")
ax[1].scatter(U_embedded[:,0], U_embedded[:,1], s=1.0)

# plt.show() doesn't seem to work so this is a workaround
def show(p):
    img = StringIO.StringIO()
    p.savefig(img, format='svg')
    img.seek(0)
    print "%html <div style='width:600px'>" + img.buf + "</div>"

#plt.legend(loc=2)
show(plt)
```

```
%pyspark
```

READY