

Tom 2 Wiki Topic ...

%pyspark

FINISHED

```
# PySpark CommonCrawl Topic Modelling
# Tom V / Paul J - 2/3/2018
```

```
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-1
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

READY

python Wikipedia Text Processing.ipynb)

```
%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')
```

```
%pyspark READY

# 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].topicDist
    return pd.Series(dict(zip(topicLabels, scores)))

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

```
%pyspark READY

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

```
%pyspark READY

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

```
%pyspark READY

# 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_pa
scores2.show(5)
```

%pyspark

READY

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

# 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", "averageTopicDis
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/doma
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
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

```
# Take a sample of PLDs for plotting in t-SNE and UMAP, and convert to Pandas dataframe
scores8=scores7.limit(10000)
df_sample = scores8.toPandas()

# t-SNE embedding
from sklearn.manifold import TSNE
import numpy as np
X=np.vstack(df_sample.averageTopicDistribution)
X_embedded = TSNE(n_components=2).fit_transform(X)
print(X_embedded.shape)
```

```

df_sample['x_tsne'] = X_embedded[:,0]
df_sample['y_tsne'] = X_embedded[:,1]

# UMAP embedding
import umap
U_embedded = umap.UMAP().fit_transform(X)
df_sample['x_umap'] = U_embedded[:,0]
df_sample['y_umap'] = U_embedded[:,1]

# Extract primary topic, for use in color scheme
df_sample['primary_topic'] = df_sample.averageTopicDistribution.apply(np.argmax)
df_sample['primary_topic_score'] = df_sample.averageTopicDistribution.apply(np.max)

# Convert back to Spark dataframe
df_sample_spark = sqlContext.createDataFrame(df_sample)
df_sample_spark.show()

```

l -9.20855769204408l	5.353137211795069l	29l 0.9010357886880672l
l corporatefitnessc...l	[1.16246367421139...l	50.717445373535156l 25.67108154296875
l 2.3035335272625375l	2.9489529987426724l	62l 0.23111564571532547l
l 1centweb.coml	[1.94489585421988...l	-57.56715393066406l 25.92642593383789
l -5.026943730782058l	2.6341051583784534l	29l 0.7100540119133284l
l mygaragestory.netl	[2.04009774069444...l	-51.19037628173828l 0.0740693137049675
l -3.402280101604503l	-0.4820181151768883l	29l 0.26711331698144136l
l gitedeville.coml	[1.18827049530001...l	-21.88416862487793l -25.539623260498047
l 4.569428845131571l	0.6412596295209377l	46l 0.26875609876099643l
l annadesogus.coml	[0.01558264822551...l	15.936853408813477l 21.140478134155273
l 1.052034302953693l	-0.3460609618451271l	29l 0.23605239678942952l
l michael-barrymore...l	[1.00132680529176...l	-17.30196762084961l -7.373234748840332
l -0.432409381629909l	-0.0637145346748731l	47l 0.31757600891167936l
l prostarcleaning.coml	[3.77903452976982...l	-26.730302810668945l -23.947370529174805
l -1.352855649440794l	-0.42328961848371544l	41l 0.21829410518358922l
l bzupages.coml	[1.08151075067157...l	-5.620166301727295l 62.51683044433594
l 5.644085827380796l	3.387334899818211l	29l 0.3132924891829208l
l denhall.coml	[5.22141585851430...l	-74.50524139404297l 19.671321868896484

```
%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"))

# Tidy up columns
udf_reverse_domain = udf(reverse_domain, StringType())
scores9=df_sample_spark.withColumn("actual_pld_rev",udf_reverse_domain("pld_rev"))
scores10=scores9.join(pld_df, pld_df.PLD==scores9.actual_pld_rev).withColumnRenamed("pld_rev", "primary_topic_score")
scores10.show(3)

```



```
abc.facebook.com
```

```

+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
|      payLevelDomain|NumHosts|      x_tsne|      y_tsne|      x_umap|
|      y_umap|primary_topic|primary_topic_score|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|      2bap.com|      1|-1.01448392868042|-3.0176773071289062| 0.4180089070827317|
| 0.2568085759153541|      29| 0.2239727598434192|
|laddictionrehabaus...|      1| 69.7011947631836| 10.383376121520996| -6.795308385975276|
|0.017603301001099064|      18| 0.4195210474391418|
|      ccfo.com|      1|7.305209159851074| 39.560585021972656|-0.8117916602499007|
| 3.05639909875979|      21| 0.3325174922165379|
+-----+-----+-----+-----+
+-----+-----+-----+-----+

```

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