2-Outbrain-UserProfiles

January 21, 2022

[1]:
```
evaluation = True

OUTPUT_BUCKET_FOLDER = "gs://akhilbucket/outbrain-click-prediction/output/"
DATA_BUCKET_FOLDER = "gs://akhilbucket/data/"
```

[2]:
```
from IPython.display import display
```

[3]:
```
from pyspark.sql.types import *
import pyspark.sql.functions as F
```

[4]:
```
from pyspark.sql import DataFrameWriter
```

[5]:
```
import numpy as np
```

[6]:
```
import math
import datetime
import time
```

[7]:
```
import random
random.seed(42)
```

[8]:
```
start_time = time.time()
```

0.1 Loading data

[9]:
```
truncate_day_from_timestamp_udf = F.udf(lambda ts: int(ts / 1000 / 60 / 60 /␣
↪24), IntegerType())
```

[10]:
```
extract_country_udf = F.udf(lambda geo: geo.strip()[:2] if geo != None else '',␣
↪StringType())
```

[11]:
```
documents_meta_schema = StructType(
                    [StructField("document_id_doc", IntegerType(), True),
                    StructField("source_id", IntegerType(), True),
                    StructField("publisher_id", IntegerType(), True),
                    StructField("publish_time", TimestampType(), True)]
                    )
```

```
documents_meta_df = spark.read.schema(documents_meta_schema).
 ↪options(header='true', inferschema='false', nullValue='\\N') \
                .csv(DATA_BUCKET_FOLDER+"documents_meta.csv") \
                .withColumn('dummyDocumentsMeta', F.lit(1)).
 ↪alias('documents_meta')
```

[12]:
```
documents_categories_schema = StructType(
                    [StructField("document_id_cat", IntegerType(), True),
                    StructField("category_id", IntegerType(), True),
                    StructField("confidence_level_cat", FloatType(), True)]
                    )

documents_categories_df = spark.read.schema(documents_categories_schema).
 ↪options(header='true', inferschema='false', nullValue='\\N') \
                .csv(DATA_BUCKET_FOLDER+"documents_categories.csv") \
                .alias('documents_categories')

documents_categories_grouped_df = documents_categories_df.
 ↪groupBy('document_id_cat') \
                                            .agg(F.collect_list('category_id').
 ↪alias('category_id_list'),
                                                 F.
 ↪collect_list('confidence_level_cat').alias('cat_confidence_level_list')) \
                                            .
 ↪withColumn('dummyDocumentsCategory', F.lit(1)) \
                                            .
 ↪alias('documents_categories_grouped')
```

[13]:
```
documents_topics_schema = StructType(
                    [StructField("document_id_top", IntegerType(), True),
                    StructField("topic_id", IntegerType(), True),
                    StructField("confidence_level_top", FloatType(), True)]
                    )

documents_topics_df = spark.read.schema(documents_topics_schema).
 ↪options(header='true', inferschema='false', nullValue='\\N') \
                .csv(DATA_BUCKET_FOLDER+"documents_topics.csv")  \
                .alias('documents_topics')

documents_topics_grouped_df = documents_topics_df.groupBy('document_id_top') \
                                            .agg(F.collect_list('topic_id').
 ↪alias('topic_id_list'),
                                                 F.
 ↪collect_list('confidence_level_top').alias('top_confidence_level_list')) \
```

```
                                                .withColumn('dummyDocumentsTopics',␣
↪F.lit(1)) \
                                                .alias('documents_topics_grouped')
```

[14]:
```
documents_entities_schema = StructType(
                    [StructField("document_id_ent", IntegerType(), True),
                    StructField("entity_id", StringType(), True),
                    StructField("confidence_level_ent", FloatType(), True)]
                    )

documents_entities_df = spark.read.schema(documents_entities_schema).
↪options(header='true', inferschema='false', nullValue='\\N') \
                .csv(DATA_BUCKET_FOLDER+"documents_entities.csv")  \
                .alias('documents_entities')

documents_entities_grouped_df = documents_entities_df.
↪groupBy('document_id_ent') \
                                                .agg(F.collect_list('entity_id').
↪alias('entity_id_list'),
                                                     F.
↪collect_list('confidence_level_ent').alias('ent_confidence_level_list')) \
                                                .
↪withColumn('dummyDocumentsEntities', F.lit(1)) \
                                                .alias('documents_entities_grouped')
```

[15]:
```
documents_df = documents_meta_df.join(documents_categories_grouped_df, on=F.
↪col("document_id_doc") == F.col("documents_categories_grouped.
↪document_id_cat"), how='left') \
                        .join(documents_topics_grouped_df, on=F.
↪col("document_id_doc") == F.col("documents_topics_grouped.document_id_top"),␣
↪how='left') \
                        .join(documents_entities_grouped_df, on=F.
↪col("document_id_doc") == F.col("documents_entities_grouped.
↪document_id_ent"), how='left') \
                        .cache()
```

[16]:
```
documents_df.count()
```

[16]:
```
2999334
```

[17]:
```
if evaluation:
    validation_set_df = spark.read.parquet(OUTPUT_BUCKET_FOLDER+"validation_set.
↪parquet") \
                    .alias('validation_set')
```

```
    validation_set_df.select('uuid_event').distinct().
↪createOrReplaceTempView('users_to_profile')
    validation_set_df.select('uuid_event','document_id_promo').distinct().
↪createOrReplaceTempView('validation_users_docs_to_ignore')

else:
    events_schema = StructType(
                    [StructField("display_id", IntegerType(), True),
                    StructField("uuid_event", StringType(), True),
                    StructField("document_id_event", IntegerType(), True),
                    StructField("timestamp_event", IntegerType(), True),
                    StructField("platform_event", IntegerType(), True),
                    StructField("geo_location_event", StringType(), True)]
                    )

    events_df = spark.read.schema(events_schema).options(header='true',␣
↪inferschema='false', nullValue='\\N') \
                    .csv(DATA_BUCKET_FOLDER+"events.csv") \
                    .withColumn('dummyEvents', F.lit(1)) \
                    .withColumn('day_event',␣
↪truncate_day_from_timestamp_udf('timestamp_event')) \
                    .withColumn('event_country',␣
↪extract_country_udf('geo_location_event')) \
                    .alias('events')

    events_df.createOrReplaceTempView('events')


    promoted_content_schema = StructType(
                        [StructField("ad_id", IntegerType(), True),
                        StructField("document_id_promo", IntegerType(), True),
                        StructField("campaign_id", IntegerType(), True),
                        StructField("advertiser_id", IntegerType(), True)]
                        )

    promoted_content_df = spark.read.schema(promoted_content_schema).
↪options(header='true', inferschema='false', nullValue='\\N') \
                    .csv(DATA_BUCKET_FOLDER+"promoted_content.csv") \
                    .withColumn('dummyPromotedContent', F.lit(1)).
↪alias('promoted_content')

    clicks_test_schema = StructType(
                        [StructField("display_id", IntegerType(), True),
                        StructField("ad_id", IntegerType(), True)]
                        )
```

```
    clicks_test_df = spark.read.schema(clicks_test_schema).
↪options(header='true', inferschema='false', nullValue='\\N') \
                    .csv(DATA_BUCKET_FOLDER+"clicks_test.csv") \
                    .withColumn('dummyClicksTest', F.lit(1)).
↪alias('clicks_test')

    test_set_df = clicks_test_df.join(promoted_content_df, on='ad_id',␣
↪how='left') \
                                .join(events_df, on='display_id', how='left')

    test_set_df.select('uuid_event').distinct().
↪createOrReplaceTempView('users_to_profile')
    test_set_df.select('uuid_event','document_id_promo', 'timestamp_event').
↪distinct().createOrReplaceTempView('test_users_docs_timestamp_to_ignore')
```

[20]:
```
page_views_schema = StructType(
                    [StructField("uuid_pv", StringType(), True),
                    StructField("document_id_pv", IntegerType(), True),
                    StructField("timestamp_pv", IntegerType(), True),
                    StructField("platform_pv", IntegerType(), True),
                    StructField("geo_location_pv", StringType(), True),
                    StructField("traffic_source_pv", IntegerType(), True)]
                    )
page_views_df = spark.read.schema(page_views_schema).options(header='true',␣
↪inferschema='false', nullValue='\\N') \
                .csv(DATA_BUCKET_FOLDER+"page_views_sample.csv") \
                .alias('page_views')

page_views_df.createOrReplaceTempView('page_views')
```

[21]:
```
additional_filter = ''
if evaluation:
    additional_filter = '''
                            AND NOT EXISTS (SELECT uuid_event FROM␣
↪validation_users_docs_to_ignore
                                                    WHERE uuid_event = p.
↪uuid_pv
                                                   AND document_id_promo = p.
↪document_id_pv)
                        '''
else:
    additional_filter = '''
```

```
                                AND NOT EXISTS (SELECT uuid_event FROM␣
↪test_users_docs_timestamp_to_ignore
                                                 WHERE uuid_event = p.
↪uuid_pv
                                                AND document_id_promo = p.
↪document_id_pv
                                                AND p.timestamp_pv >=␣
↪timestamp_event)
                           '''

page_views_train_df = spark.sql('''SELECT * FROM page_views p
                                    WHERE EXISTS (SELECT uuid_event FROM␣
↪users_to_profile
                                                  WHERE uuid_event = p.uuid_pv) ␣
↪
                                 '''+ additional_filter
                                ).alias('views') \
                            .join(documents_df, on=F.col("document_id_pv") == F.
↪col("document_id_doc"), how='left') \
                            .filter('dummyDocumentsEntities is not null OR␣
↪dummyDocumentsTopics is not null OR dummyDocumentsCategory is not null')
```

0.2 Processing document frequencies

[22]:
```
import pickle
```

[23]:
```
documents_total = documents_meta_df.count()
documents_total
```

[23]: 2999334

[24]:
```
categories_docs_counts = documents_categories_df.groupBy('category_id').count().
↪rdd.collectAsMap()
len(categories_docs_counts)
```

[24]: 97

[25]:
```
df_filenames_suffix = ''
if evaluation:
    df_filenames_suffix = '_eval'
```

[26]:
```
with open('categories_docs_counts'+df_filenames_suffix+'.pickle', 'wb') as␣
↪output:
    pickle.dump(categories_docs_counts, output)
```

[27]:
```
topics_docs_counts = documents_topics_df.groupBy('topic_id').count().rdd.
↪collectAsMap()
len(topics_docs_counts)
```

[27]:
```
300
```

[28]:
```
with open('topics_docs_counts'+df_filenames_suffix+'.pickle', 'wb') as output:
    pickle.dump(topics_docs_counts, output)
```

[29]:
```
entities_docs_counts = documents_entities_df.groupBy('entity_id').count().rdd.
↪collectAsMap()
len(entities_docs_counts)
```

[29]:
```
1326009
```

[30]:
```
with open('entities_docs_counts'+df_filenames_suffix+'.pickle', 'wb') as output:
    pickle.dump(entities_docs_counts, output)
```

0.3 Processing user profiles

[31]:
```
int_null_to_minus_one_udf = F.udf(lambda x: x if x != None else -1,␣
↪IntegerType())
int_list_null_to_empty_list_udf = F.udf(lambda x: x if x != None else [],␣
↪ArrayType(IntegerType()))
float_list_null_to_empty_list_udf = F.udf(lambda x: x if x != None else [],␣
↪ArrayType(FloatType()))
str_list_null_to_empty_list_udf = F.udf(lambda x: x if x != None else [],␣
↪ArrayType(StringType()))
```

[32]:
```
page_views_by_user_df = page_views_train_df.select(
                            'uuid_pv',
                            'document_id_pv',
                            int_null_to_minus_one_udf('timestamp_pv').
↪alias('timestamp_pv'),
                            int_list_null_to_empty_list_udf('category_id_list').
↪alias('category_id_list'),
```

```
                          ␣
↪float_list_null_to_empty_list_udf('cat_confidence_level_list').
↪alias('cat_confidence_level_list'),
                           int_list_null_to_empty_list_udf('topic_id_list').
↪alias('topic_id_list'),
                          ␣
↪float_list_null_to_empty_list_udf('top_confidence_level_list').
↪alias('top_confidence_level_list'),
                           str_list_null_to_empty_list_udf('entity_id_list').
↪alias('entity_id_list'),
                          ␣
↪float_list_null_to_empty_list_udf('ent_confidence_level_list').
↪alias('ent_confidence_level_list')) \
                  .groupBy('uuid_pv') \
                  .agg(F.collect_list('document_id_pv').
↪alias('document_id_pv_list'),
                       F.collect_list('timestamp_pv').
↪alias('timestamp_pv_list'),
                       F.collect_list('category_id_list').
↪alias('category_id_lists'),
                       F.collect_list('cat_confidence_level_list').
↪alias('cat_confidence_level_lists'),
                       F.collect_list('topic_id_list').
↪alias('topic_id_lists'),
                       F.collect_list('top_confidence_level_list').
↪alias('top_confidence_level_lists'),
                       F.collect_list('entity_id_list').
↪alias('entity_id_lists'),
                       F.collect_list('ent_confidence_level_list').
↪alias('ent_confidence_level_lists')
                      )
```

[33]:
```
from collections import defaultdict

def get_user_aspects(docs_aspects, aspect_docs_counts):
    docs_aspects_merged_lists = defaultdict(list)

    for doc_aspects in docs_aspects:
        for key in doc_aspects.keys():
            docs_aspects_merged_lists[key].append(doc_aspects[key])

    docs_aspects_stats = {}
    for key in docs_aspects_merged_lists.keys():
        aspect_list = docs_aspects_merged_lists[key]
        tf = len(aspect_list)
        idf = math.log(documents_total / float(aspect_docs_counts[key]))
```

```
        confid_mean = sum(aspect_list) / float(len(aspect_list))
        docs_aspects_stats[key] = [tf*idf, confid_mean]


    return docs_aspects_stats


def generate_user_profile(docs_aspects_list, docs_aspects_confidence_list,␣
↪aspect_docs_counts):
    docs_aspects = []
    for doc_aspects_list, doc_aspects_confidence_list in zip(docs_aspects_list,␣
↪docs_aspects_confidence_list):
        doc_aspects = dict(zip(doc_aspects_list, doc_aspects_confidence_list))
        docs_aspects.append(doc_aspects)

    user_aspects = get_user_aspects(docs_aspects, aspect_docs_counts)
    return user_aspects
```

[34]:
```
get_list_len_udf = F.udf(lambda docs_list: len(docs_list), IntegerType())
```

[35]:
```
generate_categories_user_profile_map_udf = F.udf(lambda docs_aspects_list,
                                                 docs_aspects_confidence_list: \
                                                     ␣
↪generate_user_profile(docs_aspects_list,
                                                                              ␣
↪docs_aspects_confidence_list,
                                                                              ␣
↪categories_docs_counts),
                                          MapType(IntegerType(),
                                                  ArrayType(FloatType()),
                                                  False))


generate_topics_user_profile_map_udf = F.udf(lambda docs_aspects_list,
                                                 docs_aspects_confidence_list: \
                                                     ␣
↪generate_user_profile(docs_aspects_list,
                                                                              ␣
↪docs_aspects_confidence_list,
                                                                              ␣
↪topics_docs_counts),
                                          MapType(IntegerType(),
                                                  ArrayType(FloatType()),
                                                  False))
```

```
generate_entities_user_profile_map_udf = F.udf(lambda docs_aspects_list,
                                               docs_aspects_confidence_list: \
                                                   ␣
 ↪generate_user_profile(docs_aspects_list,
                                                                              ␣
 ↪docs_aspects_confidence_list,
                                                                              ␣
 ↪entities_docs_counts),
                                          MapType(StringType(),
                                                  ArrayType(FloatType()),
                                                  False))
```

[36]:
```
users_profile_df = page_views_by_user_df \
                                .withColumn('views',␣
 ↪get_list_len_udf('document_id_pv_list')) \
                                .withColumn('categories',
                                            ␣
 ↪generate_categories_user_profile_map_udf('category_id_lists',
                                                                      ␣
 ↪'cat_confidence_level_lists')) \
                                .withColumn('topics',
                                            ␣
 ↪generate_topics_user_profile_map_udf('topic_id_lists',
                                                                           ␣
 ↪'top_confidence_level_lists')) \
                                .withColumn('entities',
                                            ␣
 ↪generate_entities_user_profile_map_udf('entity_id_lists',
                                                                           ␣
 ↪'ent_confidence_level_lists')) \
                                .select(F.col('uuid_pv').alias('uuid'),
                                        F.col('document_id_pv_list').
 ↪alias('doc_ids'),
                                        'views',
                                        'categories', 'topics', 'entities')
```

[37]:
```
if evaluation:
    table_name = 'user_profiles_eval'
else:
    table_name = 'user_profiles'


users_profile_df.write.parquet(OUTPUT_BUCKET_FOLDER+table_name,␣
 ↪mode='overwrite')
```

[38]:
```
finish_time = time.time()
print("Elapsed min: ", (finish_time-start_time)/60/60)
```

```
Elapsed min:  0.23941108153926
```