3-Outbrain-Preprocessing

January 21, 2022

[1]:
```
evaluation = True
evaluation_verbose = False

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
from pyspark.ml.linalg import Vectors, SparseVector, VectorUDT
```

[4]:
```
import numpy as np
import scipy.sparse
```

[5]:
```
import math
import datetime
import time
import itertools
```

[6]:
```
import pickle
```

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

[8]:
```
import pandas as pd
%matplotlib inline
```

[9]:
```
start_time = time.time()
```

[10]:
```
import hashlib
def hashstr(s, nr_bins):
    return int(hashlib.md5(s.encode('utf8')).hexdigest(), 16)%(nr_bins-1)+1
```

0.1 UDFs

[11]:
```
def date_time_to_unix_epoch(date_time):
    return int(time.mktime(date_time.timetuple()))

def date_time_to_unix_epoch_treated(dt):
    if dt != None:
        try:
            epoch = date_time_to_unix_epoch(dt)
            return epoch
        except Exception as e:
            print("Error processing dt={}".format(dt), e)
            return 0
    else:
        return 0
```

[12]:
```
timestamp_null_to_zero_int_udf = F.udf(lambda x:␣
↪date_time_to_unix_epoch_treated(x), IntegerType())
```

[13]:
```
INT_DEFAULT_NULL_VALUE = -1
int_null_to_minus_one_udf = F.udf(lambda x: x if x != None else␣
↪INT_DEFAULT_NULL_VALUE, 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()))
```

[14]:
```
def truncate_day_from_timestamp(ts):
    return int(ts / 1000 / 60 / 60 / 24)
```

[15]:
```
truncate_day_from_timestamp_udf = F.udf(lambda ts:␣
↪truncate_day_from_timestamp(ts), IntegerType())
```

[16]:
```
extract_country_udf = F.udf(lambda geo: geo.strip()[:2] if geo != None else '',␣
↪StringType())
```

[17]:
```
extract_country_state_udf = F.udf(lambda geo: geo.strip()[:5] if geo != None␣
↪else '', StringType())
```

[18]:
```
list_len_udf = F.udf(lambda x: len(x) if x != None else 0, IntegerType())
```

[19]:
```
def convert_odd_timestamp(timestamp_ms_relative):
    TIMESTAMP_DELTA=1465876799998
    return datetime.datetime.
↪fromtimestamp((int(timestamp_ms_relative)+TIMESTAMP_DELTA)//1000)
```

1 Loading Files

1.1 Loading UTC/BST for each country and US / CA states (local time)

[20]:
```
country_utc_dst_df = pd.
↪read_csv(DATA_BUCKET_FOLDER+'country_codes_utc_dst_tz_delta.csv',␣
↪keep_default_na=False)
```

[21]:
```
countries_utc_dst_dict = dict(zip(country_utc_dst_df['country_code'].tolist(),␣
↪country_utc_dst_df['utc_dst_time_offset_cleaned'].tolist()))
countries_utc_dst_broad = sc.broadcast(countries_utc_dst_dict)
```

[22]:
```
us_states_utc_dst_df = pd.read_csv(DATA_BUCKET_FOLDER+'us_states_abbrev_bst.
↪csv', keep_default_na=False)
```

[23]:
```
us_states_utc_dst_dict = dict(zip(us_states_utc_dst_df['state_abb'].tolist(),␣
↪us_states_utc_dst_df['utc_dst_time_offset_cleaned'].tolist()))
us_states_utc_dst_broad = sc.broadcast(us_states_utc_dst_dict)
```

[24]:
```
ca_states_utc_dst_df = pd.read_csv(DATA_BUCKET_FOLDER+'ca_states_abbrev_bst.
↪csv', keep_default_na=False)
```

[25]:
```
ca_countries_utc_dst_dict = dict(zip(ca_states_utc_dst_df['state_abb'].
↪tolist(), ca_states_utc_dst_df['utc_dst_time_offset_cleaned'].tolist()))
ca_countries_utc_dst_broad = sc.broadcast(ca_countries_utc_dst_dict)
```

1.2 Loading competition csvs

[26]:
```
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')) \
                .withColumn('event_country_state',␣
↪extract_country_state_udf('geo_location_event')) \
```

```
                .alias('events')
```

[27]:
```
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") \
                .withColumn('day_pv',␣
 ↪truncate_day_from_timestamp_udf('timestamp_pv')) \
                .alias('page_views')

page_views_df.createOrReplaceTempView('page_views')
```

[28]:
```
page_views_users_df  = spark.sql('''
                    SELECT uuid_pv, document_id_pv, max(timestamp_pv) as␣
 ↪max_timestamp_pv, 1 as dummyPageView
                    FROM page_views p
                    GROUP BY uuid_pv, document_id_pv
                    ''').alias('page_views_users')
```

[29]:
```
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').cache()
```

[30]:
```
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').cache()
```

[31]:
```
#Joining with Page Views to get traffic_source_pv
events_joined_df = events_df.join(documents_meta_df \
                                  .withColumnRenamed('source_id',␣
 ↪'source_id_doc_event') \
                                  .withColumnRenamed('publisher_id',␣
 ↪'publisher_doc_event') \
                                  .withColumnRenamed('publish_time',␣
 ↪'publish_time_doc_event')
                                  , on=F.col("document_id_event") == F.
 ↪col("document_id_doc"), how='left') \
                            .join(page_views_df,
                                        on=[F.col('uuid_event') == F.
 ↪col('uuid_pv'),
                                            F.col('document_id_event') == F.
 ↪col('document_id_pv'),
                                            F.col('platform_event') == F.
 ↪col('platform_pv'),
                                            F.col('geo_location_event') == F.
 ↪col('geo_location_pv'),
                                            F.col('day_event') == F.
 ↪col('day_pv')],
                                       how='left') \
                                  .alias('events').cache()
```

[32]:
```
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').cache()

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('confidence_level_cat_list')) \
                                               .
↪withColumn('dummyDocumentsCategory', F.lit(1)) \
                                               .
↪alias('documents_categories_grouped')
```

[33]:
```
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').cache()

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

[34]:
```
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').cache()

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('confidence_level_ent_list')) \
                                               .
↪withColumn('dummyDocumentsEntities', F.lit(1)) \
```

```
                                                    .alias('documents_entities_grouped')
```

[35]:
```
clicks_train_schema = StructType(
                     [StructField("display_id", IntegerType(), True),
                     StructField("ad_id", IntegerType(), True),
                     StructField("clicked", IntegerType(), True)]
                     )

clicks_train_df = spark.read.schema(clicks_train_schema).options(header='true',␣
↪inferschema='false', nullValue='\\N') \
                .csv(DATA_BUCKET_FOLDER+"clicks_train.csv") \
                .withColumn('dummyClicksTrain', F.lit(1)).alias('clicks_train')
```

[36]:
```
clicks_train_joined_df = clicks_train_df \
                         .join(promoted_content_df, on='ad_id', how='left') \
                         .join(documents_meta_df, on=F.col("promoted_content.
↪document_id_promo") == F.col("documents_meta.document_id_doc"), how='left') \
                         .join(events_joined_df, on='display_id', how='left')
clicks_train_joined_df.createOrReplaceTempView('clicks_train_joined')
```

```
22/01/15 23:38:40 WARN org.apache.spark.sql.catalyst.util.package: Truncated the
string representation of a plan since it was too large. This behavior can be
adjusted by setting 'spark.sql.debug.maxToStringFields'.
```

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

user_profiles_df = spark.read.parquet(OUTPUT_BUCKET_FOLDER+table_name) \
                    .withColumn('dummyUserProfiles', F.lit(1)).
↪alias('user_profiles')
```

2 Spliting Train/validation set | Test set

[38]:
```
if evaluation:
    validation_set_exported_df = spark.read.
↪parquet(OUTPUT_BUCKET_FOLDER+"validation_set.parquet") \
                    .alias('validation_set')

    validation_set_exported_df.select('display_id').distinct().
↪createOrReplaceTempView("validation_display_ids")
```

```
    validation_set_df = spark.sql('''SELECT * FROM clicks_train_joined t
             WHERE EXISTS (SELECT display_id FROM validation_display_ids
                           WHERE display_id = t.display_id)''').alias('clicks')␣
↪\
                          .join(documents_categories_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_categories_grouped.
↪document_id_cat"), how='left') \
                          .join(documents_topics_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_topics_grouped.
↪document_id_top"), how='left') \
                          .join(documents_entities_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_entities_grouped.
↪document_id_ent"), how='left') \
                          .join(documents_categories_grouped_df \
                                    .withColumnRenamed('category_id_list',␣
↪'doc_event_category_id_list')
                                    .
↪withColumnRenamed('confidence_level_cat_list',␣
↪'doc_event_confidence_level_cat_list') \
                                    .alias('documents_event_categories_grouped'),
                                on=F.col("document_id_event") == F.
↪col("documents_event_categories_grouped.document_id_cat"),
                                how='left') \
                          .join(documents_topics_grouped_df \
                                    .withColumnRenamed('topic_id_list',␣
↪'doc_event_topic_id_list')
                                    .
↪withColumnRenamed('confidence_level_top_list',␣
↪'doc_event_confidence_level_top_list') \
                                    .alias('documents_event_topics_grouped'),
                                on=F.col("document_id_event") == F.
↪col("documents_event_topics_grouped.document_id_top"),
                                how='left') \
                          .join(documents_entities_grouped_df \
                                    .withColumnRenamed('entity_id_list',␣
↪'doc_event_entity_id_list')
                                    .
↪withColumnRenamed('confidence_level_ent_list',␣
↪'doc_event_confidence_level_ent_list') \
                                    .alias('documents_event_entities_grouped'),
                                on=F.col("document_id_event") == F.
↪col("documents_event_entities_grouped.document_id_ent"),
                                how='left') \
                          .join(page_views_users_df, on=[F.col("clicks.
↪uuid_event") == F.col("page_views_users.uuid_pv"),
```

```
                                                    F.col("clicks.
↪document_id_promo") == F.col("page_views_users.document_id_pv")],
                                                   how='left')

    #print("validation_set_df.count() =", validation_set_df.count())

    #Added to validation set information about the event and the user for␣
↪statistics of the error (avg ctr)
    validation_set_ground_truth_df = validation_set_df.filter('clicked = 1') \
                                .join(user_profiles_df, on=[F.
↪col("user_profiles.uuid") == F.col("uuid_event")], how='left') \
                                .withColumn('user_categories_count',␣
↪list_len_udf('category_id_list')) \
                                .withColumn('user_topics_count',␣
↪list_len_udf('topic_id_list')) \
                                .withColumn('user_entities_count',␣
↪list_len_udf('entity_id_list')) \
                                .select('display_id','ad_id','platform_event',␣
↪'day_event', 'timestamp_event',
                                        'geo_location_event', 'event_country',␣
↪'event_country_state', 'views',
                                        'user_categories_count',␣
↪'user_topics_count', 'user_entities_count') \
                                .withColumnRenamed('ad_id','ad_id_gt') \
                                .withColumnRenamed('views','user_views_count') \
                                .cache()
    #print("validation_set_ground_truth_df.count() =",␣
↪validation_set_ground_truth_df.count())

    train_set_df = spark.sql('''SELECT * FROM clicks_train_joined t
                                WHERE NOT EXISTS (SELECT display_id FROM␣
↪validation_display_ids
                                                  WHERE display_id = t.
↪display_id)''').cache()
    print("train_set_df.count() =", train_set_df.count())

    #validation_display_ids_df.groupBy("day_event").count().show()

else:

    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)) \
                    .withColumn('clicked', F.lit(-999)) \
                    .alias('clicks_test')


   test_set_df = clicks_test_df \
                          .join(promoted_content_df, on='ad_id', how='left') \
                          .join(documents_meta_df, on=F.col("promoted_content.
↪document_id_promo") == F.col("documents_meta.document_id_doc"), how='left') \
                          .join(documents_categories_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_categories_grouped.
↪document_id_cat"), how='left') \
                          .join(documents_topics_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_topics_grouped.
↪document_id_top"), how='left') \
                          .join(documents_entities_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_entities_grouped.
↪document_id_ent"), how='left') \
                          .join(events_joined_df, on='display_id', how='left') \
                          .join(documents_categories_grouped_df \
                                    .withColumnRenamed('category_id_list',␣
↪'doc_event_category_id_list')
                                    .
↪withColumnRenamed('confidence_level_cat_list',␣
↪'doc_event_confidence_level_cat_list') \
                                    .alias('documents_event_categories_grouped'),
                                on=F.col("document_id_event") == F.
↪col("documents_event_categories_grouped.document_id_cat"),
                                how='left') \
                          .join(documents_topics_grouped_df \
                                    .withColumnRenamed('topic_id_list',␣
↪'doc_event_topic_id_list')
                                    .
↪withColumnRenamed('confidence_level_top_list',␣
↪'doc_event_confidence_level_top_list') \
                                    .alias('documents_event_topics_grouped'),
                                on=F.col("document_id_event") == F.
↪col("documents_event_topics_grouped.document_id_top"),
                                how='left') \
                          .join(documents_entities_grouped_df \
                                    .withColumnRenamed('entity_id_list',␣
↪'doc_event_entity_id_list')
```

```
                                    .
↪withColumnRenamed('confidence_level_ent_list',␣
↪'doc_event_confidence_level_ent_list') \
                                        .alias('documents_event_entities_grouped'),
                                on=F.col("document_id_event") == F.
↪col("documents_event_entities_grouped.document_id_ent"),
                                how='left') \
                            .join(page_views_users_df, on=[F.col("events.
↪uuid_event") == F.col("page_views_users.uuid_pv"),
                                                           F.col("promoted_content.
↪document_id_promo") == F.col("page_views_users.document_id_pv")],
                                                       how='left')

    #print("test_set_df.count() =",test_set_df.count())


    train_set_df = clicks_train_joined_df.cache()
    print("train_set_df.count() =", train_set_df.count())
```

```
[Stage 13:=====================================================>(199 + 1) / 200]
train_set_df.count() = 59776575
```

3 Training models

[39]:
```
def is_null(value):
    return value == None or len(str(value).strip()) == 0
```

[40]:
```
LESS_SPECIAL_CAT_VALUE = 'less'
def get_category_field_values_counts(field, df, min_threshold=10):
    category_counts = dict(list(filter(lambda x: not is_null(x[0]) and x[1] >=␣
 ↪min_threshold, df.select(field).groupBy(field).count().rdd.map(lambda x:␣
 ↪(x[0], x[1])).collect())))
    #Adding a special value to create a feature for values in this category␣
 ↪that are less than min_threshold
    category_counts[LESS_SPECIAL_CAT_VALUE] = -1
    return category_counts
```

3.1 Building category values counters and indexers

[42]:
```
event_country_values_counts = get_category_field_values_counts('event_country',␣
 ↪events_df, min_threshold=10)
len(event_country_values_counts)
#All non-null categories: 230
```

[42]: 222

```
[43]: event_country_state_values_counts =␣
      ↪get_category_field_values_counts('event_country_state', events_df,␣
      ↪min_threshold=10)
      len(event_country_state_values_counts)
```

[43]: 1892

```
[44]: event_geo_location_values_counts =␣
      ↪get_category_field_values_counts('geo_location_event', events_df,␣
      ↪min_threshold=10)
      len(event_geo_location_values_counts)
      #All non-null categories: 2988
```

[44]: 2273

```
[45]: doc_entity_id_values_counts = get_category_field_values_counts('entity_id',␣
      ↪documents_entities_df, min_threshold=10)
      len(doc_entity_id_values_counts)
      #All non-null categories: 1326009
```

[45]: 52439

3.2 Processing average CTR by categories

```
[46]: def get_percentiles(df, field, quantiles_levels=None, max_error_rate=0.0):
          if quantiles_levels == None:
              quantiles_levels = np.arange(0.0, 1.1, 0.1).tolist()
          quantiles = df.approxQuantile(field, quantiles_levels, max_error_rate)
          return dict(zip(quantiles_levels, quantiles))
```

```
[47]: #REG = 10
      REG = 0
      ctr_udf = F.udf(lambda clicks, views: clicks / float(views + REG), FloatType())
```

3.2.1 Average CTR by ad_id

[48]:
```
ad_id_popularity_df = train_set_df.groupby('ad_id').agg(F.sum('clicked').
↪alias('clicks'),
                                                        F.count('*').
↪alias('views')) \
                                            .withColumn('ctr',␣
↪ctr_udf('clicks','views'))
```

[49]:
```
#ad_id_popularity_df.count()
```

[50]:
```
#get_percentiles(ad_id_popularity_df, 'clicks')
```

[51]:
```
#get_percentiles(ad_id_popularity_df, 'views')
```

[52]:
```
ad_id_popularity = ad_id_popularity_df.filter('views > 5').select('ad_id',␣
↪'ctr', 'views') \
                    .rdd.map(lambda x: (x['ad_id'], (x['ctr'], x['views'], 1,␣
↪1))).collectAsMap()
```

[53]:
```
ad_id_popularity_broad = sc.broadcast(ad_id_popularity)
```

[54]:
```
list(ad_id_popularity.values())[:3]
```

[54]:
```
[(0.3709402084350586, 33407, 1, 1),
 (0.10940171033143997, 585, 1, 1),
 (0.09989330172538757, 7498, 1, 1)]
```

[55]:
```
len(ad_id_popularity)
```

[55]:
```
192233
```

[56]:
```
#get_ad_id_ctr_udf = F.udf(lambda ad_id: ad_id_popularity[ad_id] if ad_id in␣
↪ad_id_popularity else -1, FloatType())
```

[57]:
```
ad_id_avg_ctr = sum(map(lambda x: x[0], ad_id_popularity.values())) /␣
↪float(len(ad_id_popularity))
ad_id_avg_ctr
```

[57]:
```
0.15546540640749865
```

[58]:
```
ad_id_weighted_avg_ctr = sum(map(lambda x: x[0]*x[1], ad_id_popularity.
↪values())) / float(sum(map(lambda x: x[1], ad_id_popularity.values())))
ad_id_weighted_avg_ctr
```

[58]:
```
0.19403065845149367
```

[59]:
```
ad_id_views_median = np.median(np.array(list(map(lambda x: x[1],␣
↪ad_id_popularity.values()))))
ad_id_views_median
```

[59]:
```
18.0
```

[60]:
```
ad_id_views_mean = sum(map(lambda x: x[1], ad_id_popularity.values())) /␣
↪float(len(ad_id_popularity))
ad_id_views_mean
```

[60]:
```
308.4728688622661
```

3.2.2 Average CTR by document_id (promoted_content)

[61]:
```
document_id_popularity_df = train_set_df.groupby('document_id_promo').agg(F.
↪sum('clicked').alias('clicks'),
                                                                    F.count('*').
↪alias('views'),
                                                                    F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                            .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

document_id_popularity = document_id_popularity_df.filter('views > 5').
↪select('document_id_promo', 'ctr', 'views', 'distinct_ad_ids') \
                                                .rdd.map(lambda x:␣
↪(x['document_id_promo'], (x['ctr'], x['views'], x['distinct_ad_ids'], 1))).
↪collectAsMap()
len(document_id_popularity)
```

[61]:
```
74733
```

[62]:
```
document_id_popularity_broad = sc.broadcast(document_id_popularity)
```

[63]:
```
#document_id_popularity_df.count()
```

[64]:
```
#get_percentiles(document_id_popularity_df, 'clicks')
```

[65]:
```
#get_percentiles(document_id_popularity_df, 'views')
```

[66]:
```
document_id_avg_ctr = sum(map(lambda x: x[0], document_id_popularity.values()))␣
↪/ float(len(document_id_popularity))
```

```
document_id_avg_ctr
```

[66]: 0.15071710517408332

[67]:
```
document_id_weighted_avg_ctr = sum(list(map(lambda x: x[0]*x[1],␣
↪document_id_popularity.values()))) / float(sum(list(map(lambda x: x[1],␣
↪document_id_popularity.values()))))
document_id_weighted_avg_ctr
```

[67]: 0.19378523996640107

[68]:
```
document_id_views_median = np.median(np.array(list(map(lambda x: x[1],␣
↪document_id_popularity.values()))))
document_id_views_median
```

[68]: 28.0

[69]:
```
document_id_views_mean = sum(map(lambda x: x[1], document_id_popularity.
↪values())) / float(len(document_id_popularity))
document_id_views_mean
```

[69]: 797.9473458846828

3.2.3 Average CTR by (doc_event, doc_ad)

[70]:
```
doc_event_doc_ad_avg_ctr_df = train_set_df.groupBy('document_id_event',␣
↪'document_id_promo') \
                                    .agg(F.sum('clicked').alias('clicks'),
                                         F.count('*').alias('views'),
                                         F.countDistinct('ad_id').
↪alias('distinct_ad_ids')) \
                                    .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

doc_event_doc_ad_avg_ctr = doc_event_doc_ad_avg_ctr_df.filter('views > 5') \
                    .select('document_id_event', 'document_id_promo','ctr',␣
↪'views', 'distinct_ad_ids') \
                    .rdd.map(lambda x: ((x['document_id_event'],␣
↪x['document_id_promo']), (x['ctr'], x['views'], x['distinct_ad_ids'], 1))).
↪collectAsMap()

len(doc_event_doc_ad_avg_ctr)
```

[70]: 1302594

[71]:
```
doc_event_doc_ad_avg_ctr_broad = sc.broadcast(doc_event_doc_ad_avg_ctr)
```

3.2.4 Average CTR by country, source_id

[72]:
```
source_id_by_country_popularity_df = train_set_df.select('clicked',␣
↪'source_id', 'event_country', 'ad_id') \
                                                .groupby('event_country',␣
↪'source_id').agg(F.sum('clicked').alias('clicks'),
                                                                        F.
↪count('*').alias('views'),
                                                                        F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                               .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

#source_id_popularity = source_id_popularity_df.filter('views > 100 and␣
↪source_id is not null').select('source_id', 'ctr').rdd.collectAsMap()
source_id_by_country_popularity = source_id_by_country_popularity_df.
↪filter('views > 5 and source_id is not null and event_country <> ""').
↪select('event_country', 'source_id', 'ctr', 'views', 'distinct_ad_ids') \
       .rdd.map(lambda x: ((x['event_country'], x['source_id']), (x['ctr'],␣
↪x['views'], x['distinct_ad_ids'], 1))).collectAsMap()
len(source_id_by_country_popularity)
```

[72]:
```
29866
```

[73]:
```
source_id_by_country_popularity_broad = sc.
↪broadcast(source_id_by_country_popularity)
```

[74]:
```
source_id_by_country_avg_ctr = sum(map(lambda x: x[0],␣
↪source_id_by_country_popularity.values())) /␣
↪float(len(source_id_by_country_popularity))
source_id_by_country_avg_ctr
```

[74]:
```
0.18542310673302428
```

[75]:
```
source_id_by_country_weighted_avg_ctr = sum(map(lambda x: x[0]*x[1],␣
↪source_id_by_country_popularity.values())) / float(sum(map(lambda x: x[1],␣
↪source_id_by_country_popularity.values())))
source_id_by_country_weighted_avg_ctr
```

[75]:
```
0.19362581384947233
```

[76]:
```
source_id_by_country_views_median = np.median(np.array(list(map(lambda x: x[1],␣
↪source_id_by_country_popularity.values()))))
source_id_by_country_views_median
```

[76]:
```
38.0
```

[77]:
```
source_id_by_country_views_mean = sum(map(lambda x: x[1],␣
↪source_id_by_country_popularity.values())) /␣
↪float(len(source_id_by_country_popularity))
source_id_by_country_views_mean
```

[77]:
```
1998.9951114980245
```

3.2.5 Average CTR by source_id

[78]:
```
source_id_popularity_df = train_set_df.select('clicked', 'source_id', 'ad_id') \
                                            .groupby('source_id').agg(F.
↪sum('clicked').alias('clicks'),
                                                                      F.
↪count('*').alias('views'),
                                                                      F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                          .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

source_id_popularity = source_id_popularity_df.filter('views > 10 and source_id␣
↪is not null').select('source_id', 'ctr', 'views', 'distinct_ad_ids') \
                            .rdd.map(lambda x: (x['source_id'], (x['ctr'],␣
↪x['views'], x['distinct_ad_ids'], 1))).collectAsMap()
len(source_id_popularity)
```

[78]:
```
5640
```

[79]:
```
source_id_popularity_broad = sc.broadcast(source_id_popularity)
```

[80]:
```
#source_id_popularity_df.count()
```

[81]:
```
#get_percentiles(source_id_popularity_df, 'clicks')
```

[82]:
```
#get_percentiles(source_id_popularity_df, 'views')
```

[83]:
```
#source_id_popularity = source_id_popularity_df.filter('views > 100 and␣
↪source_id is not null').select('source_id', 'ctr').rdd.collectAsMap()
```

3.2.6 Average CTR by publisher_id

[84]:
```
publisher_popularity_df = train_set_df.select('clicked', 'publisher_id',␣
↪'ad_id') \
                                                .groupby('publisher_id').agg(F.
↪sum('clicked').alias('clicks'),
                                                                             F.
↪count('*').alias('views'),
                                                                              F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                               .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

publisher_popularity = publisher_popularity_df.filter('views > 10 and␣
↪publisher_id is not null').select('publisher_id', 'ctr', 'views',␣
↪'distinct_ad_ids') \
                               .rdd.map(lambda x: (x['publisher_id'], (x['ctr'],␣
↪x['views'], x['distinct_ad_ids'], 1))).collectAsMap()
len(publisher_popularity)
```

[84]:
```
724
```

[85]:
```
publisher_popularity_broad = sc.broadcast(publisher_popularity)
```

[86]:
```
#publisher_popularity_df.count()
##863
```

[87]:
```
#get_percentiles(publisher_popularity_df, 'clicks')
```

[88]:
```
#get_percentiles(publisher_popularity_df, 'views')
```

[89]:
```
#publisher_id_popularity = publisher_popularity_df.filter('views > 100 and␣
↪publisher_id is not null').select('publisher_id', 'ctr').rdd.collectAsMap()
#len(publisher_id_popularity)
##639
```

3.2.7 Average CTR by advertiser_id

[90]:
```
advertiser_id_popularity_df = train_set_df.select('clicked', 'advertiser_id',␣
↪'ad_id') \
                                                .groupby('advertiser_id').agg(F.
↪sum('clicked').alias('clicks'),
                                                                              F.
↪count('*').alias('views'),
```

```
                                                                    F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                          .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

advertiser_id_popularity = advertiser_id_popularity_df.filter('views > 10 and␣
↪advertiser_id is not null').select('advertiser_id', 'ctr', 'views',␣
↪'distinct_ad_ids') \
                            .rdd.map(lambda x: (x['advertiser_id'], (x['ctr'],␣
↪x['views'], x['distinct_ad_ids'], 1))).collectAsMap()
len(advertiser_id_popularity)
```

[90]: 3627

[91]:
```
advertiser_id_popularity_broad = sc.broadcast(advertiser_id_popularity)
```

[92]:
```
#advertiser_id_popularity_df.count()
##4063
```

[93]:
```
#get_percentiles(advertiser_id_popularity_df, 'clicks')
```

[94]:
```
#get_percentiles(advertiser_id_popularity_df, 'views')
```

[95]:
```
#advertiser_id_popularity = advertiser_id_popularity_df.filter('views > 100 and␣
↪advertiser_id is not null').select('advertiser_id', 'ctr').rdd.collectAsMap()
#len(advertiser_id_popularity)
##3129
```

3.2.8 Average CTR by campaign_id

[96]:
```
campaign_id_popularity_df = train_set_df.select('clicked', 'campaign_id',␣
↪'ad_id') \
                                        .groupby('campaign_id').agg(F.
↪sum('clicked').alias('clicks'),
                                                                    F.
↪count('*').alias('views'),
                                                                    F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                          .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

campaign_id_popularity = campaign_id_popularity_df.filter('views > 10 and␣
↪campaign_id is not null').select('campaign_id', 'ctr', 'views',␣
↪'distinct_ad_ids') \
```

```
                                .rdd.map(lambda x: (x['campaign_id'], (x['ctr'],␣
 ↪x['views'], x['distinct_ad_ids'], 1))).collectAsMap()
len(campaign_id_popularity)
```

[96]: 25260

[97]:
```
campaign_id_popularity_broad = sc.broadcast(campaign_id_popularity)
```

[98]:
```
#campaign_id_popularity_df.count()
##31390
```

[99]:
```
#get_percentiles(campaign_id_popularity_df, 'clicks')
```

[100]:
```
#get_percentiles(campaign_id_popularity_df, 'views')
```

[101]:
```
#campaign_id_popularity = campaign_id_popularity_df.filter('views > 100 and␣
 ↪campaign_id is not null').select('campaign_id', 'ctr').rdd.collectAsMap()
#len(campaign_id_popularity)
##16097
```

3.2.9 Average CTR by category

[102]:
```
category_id_popularity_df = train_set_df.join(documents_categories_df.
 ↪alias('cat_local'), on=F.col("document_id_promo") == F.col("cat_local.
 ↪document_id_cat"), how='inner') \
                                        .select('clicked', 'category_id',␣
 ↪'confidence_level_cat', 'ad_id') \
                                        .groupby('category_id').agg(F.
 ↪sum('clicked').alias('clicks'),
                                                                    F.
 ↪count('*').alias('views'),
                                                                    F.
 ↪mean('confidence_level_cat').alias('avg_confidence_level_cat'),
                                                                    F.
 ↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                         .withColumn('ctr',␣
 ↪ctr_udf('clicks','views'))

category_id_popularity = category_id_popularity_df.filter('views > 10').
 ↪select('category_id', 'ctr', 'views', 'avg_confidence_level_cat',␣
 ↪'distinct_ad_ids') \
                                .rdd.map(lambda x: (x['category_id'], (x['ctr'],␣
 ↪x['views'], x['distinct_ad_ids'], x['avg_confidence_level_cat']))).
 ↪collectAsMap()
```

```
len(category_id_popularity)
```

[102]: 95

[103]:
```
category_id_popularity_broad = sc.broadcast(category_id_popularity)
```

[104]:
```
list(category_id_popularity.values())[:10]
```

[104]:
```
[(0.24791406095027924, 29603, 260, 0.2640831177178419),
 (0.2693342864513397, 1864085, 15184, 0.6950170823384569),
 (0.14589421451091766, 305605, 1874, 0.38337482873363365),
 (0.2102375328540802, 2180134, 16663, 0.5191247273181029),
 (0.16713584959506989, 168378, 2000, 0.07425040699708986),
 (0.12638108432292938, 1929363, 7419, 0.6012419409144963),
 (0.21471861004829407, 1155, 48, 0.5800865800865801),
 (0.20932844281196594, 2842724, 10911, 0.3760230175412901 4),
 (0.2147572785615921, 2799919, 31154, 0.4709342193523636),
 (0.15035928785800934, 20318, 135, 0.24416173093255786)]
```

[105]:
```
np.median(np.array(list(map(lambda x: x[1], category_id_popularity.values()))))
```

[105]: 693061.0

[106]:
```
sum(map(lambda x: x[1], category_id_popularity.values())) /␣
↪float(len(category_id_popularity))
```

[106]: 1246666.2631578948

[107]:
```
#Parece haver uma hierarquia nas categorias pelo padrão dos códigos...
#category_id_popularity
```

3.2.10 Average CTR by (country, category)

[108]:
```
category_id_by_country_popularity_df = train_set_df.
↪join(documents_categories_df.alias('cat_local'), on=F.
↪col("document_id_promo") == F.col("cat_local.document_id_cat"), how='inner')␣
↪\
                                        .select('clicked', 'category_id',␣
↪'confidence_level_cat', 'event_country', 'ad_id') \
                                        .groupby('event_country','category_id').
↪agg(F.sum('clicked').alias('clicks'),
                                                                            ␣
↪    F.count('*').alias('views'),
                                                                            ␣
↪    F.mean('confidence_level_cat').alias('avg_confidence_level_cat'),
```

```
 ↪    F.countDistinct('ad_id').alias('distinct_ad_ids')) \
                                          .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

category_id_by_country_popularity = category_id_by_country_popularity_df.
↪filter('views > 10 and event_country <> ""').select('event_country',␣
↪'category_id', 'ctr', 'views', 'avg_confidence_level_cat',␣
↪'distinct_ad_ids') \
                                      .rdd.map(lambda x: ((x['event_country'],␣
↪x['category_id']), (x['ctr'], x['views'], x['distinct_ad_ids'],␣
↪x['avg_confidence_level_cat']))).collectAsMap()
len(category_id_by_country_popularity)
```

[108]: 11006

[109]:
```
category_id_by_country_popularity_broad = sc.
↪broadcast(category_id_by_country_popularity)
```

3.2.11 Average CTR by Topic

[110]:
```
topic_id_popularity_df = train_set_df.join(documents_topics_df.
↪alias('top_local'), on=F.col("document_id_promo") == F.col("top_local.
↪document_id_top"), how='inner') \
                                      .select('clicked', 'topic_id',␣
↪'confidence_level_top', 'ad_id') \
                                      .groupby('topic_id').agg(F.
↪sum('clicked').alias('clicks'),
                                                               F.count('*').
↪alias('views'),
                                                               F.
↪mean('confidence_level_top').alias('avg_confidence_level_top'),
                                                               F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                      .withColumn('ctr',␣
↪ctr_udf('clicks','views'))
topic_id_popularity = topic_id_popularity_df.filter('views > 10').
↪select('topic_id', 'ctr', 'views', 'avg_confidence_level_top',␣
↪'distinct_ad_ids') \
                             .rdd.map(lambda x: (x['topic_id'], (x['ctr'],␣
↪x['views'], x['distinct_ad_ids'], x['avg_confidence_level_top']))).
↪collectAsMap()
len(topic_id_popularity)
```

[110]:
```
300
```

[111]:
```
topic_id_popularity_broad = sc.broadcast(topic_id_popularity)
```

[112]:
```
sum(map(lambda x: x[1], topic_id_popularity.values())) /␣
↪float(len(topic_id_popularity))
```

[112]:
```
526870.9533333334
```

[113]:
```
sum(map(lambda x: x[2]*x[1], topic_id_popularity.values())) /␣
↪float(len(topic_id_popularity))
```

[113]:
```
7002301509.573334
```

3.2.12 Average CTR by (country, topic)

[114]:
```
topic_id_by_country_popularity_df = train_set_df.join(documents_topics_df.
↪alias('top_local'), on=F.col("document_id_promo") == F.col("top_local.
↪document_id_top"), how='inner') \
                                        .select('clicked', 'topic_id',␣
↪'confidence_level_top','event_country', 'ad_id') \
                                        .groupby('event_country','topic_id').
↪agg(F.sum('clicked').alias('clicks'),
                                                                        F.
↪count('*').alias('views'),
                                                                        F.
↪mean('confidence_level_top').alias('avg_confidence_level_top'),
                                                                        F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                         .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

topic_id_id_by_country_popularity = topic_id_by_country_popularity_df.
↪filter('views > 10 and event_country <> ""').select('event_country',␣
↪'topic_id', 'ctr', 'views', 'avg_confidence_level_top', 'distinct_ad_ids') \
                            .rdd.map(lambda x: ((x['event_country'],␣
↪x['topic_id']), (x['ctr'], x['views'], x['distinct_ad_ids'],␣
↪x['avg_confidence_level_top']))).collectAsMap()
len(topic_id_id_by_country_popularity)
```

[114]:
```
33039
```

[115]:
```
topic_id_id_by_country_popularity_broad = sc.
↪broadcast(topic_id_id_by_country_popularity)
```

3.2.13 Average CTR by Entity

[116]:
```
entity_id_popularity_df = train_set_df.join(documents_entities_df.
↪alias('ent_local'), on=F.col("document_id_promo") == F.col("ent_local.
↪document_id_ent"), how='inner') \
                                        .select('clicked', 'entity_id',␣
↪'confidence_level_ent', 'ad_id') \
                                        .groupby('entity_id').agg(F.
↪sum('clicked').alias('clicks'),
                                                                  F.count('*').
↪alias('views'),
                                                                  F.
↪mean('confidence_level_ent').alias('avg_confidence_level_ent'),
                                                                  F.
↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                         .withColumn('ctr',␣
↪ctr_udf('clicks','views'))

entity_id_popularity = entity_id_popularity_df.filter('views > 5').
↪select('entity_id', 'ctr', 'views', 'avg_confidence_level_ent',␣
↪'distinct_ad_ids') \
                                    .rdd.map(lambda x: (x['entity_id'],␣
↪(x['ctr'], x['views'], x['distinct_ad_ids'],␣
↪x['avg_confidence_level_ent']))).collectAsMap()
len(entity_id_popularity)
```

[116]:
```
78097
```

[117]:
```
entity_id_popularity_broad = sc.broadcast(entity_id_popularity)
```

[118]:
```
np.median(np.array(list(map(lambda x: x[1], entity_id_popularity.values()))))
```

[118]:
```
48.0
```

[119]:
```
sum(map(lambda x: x[1], entity_id_popularity.values())) /␣
↪float(len(entity_id_popularity))
```

[119]:
```
1917.1555757583517
```

3.2.14 Average CTR by (country, entity)

[120]:
```
entity_id_by_country_popularity_df = train_set_df.join(documents_entities_df.
 ↪alias('ent_local'), on=F.col("document_id_promo") == F.col("ent_local.
 ↪document_id_ent"), how='inner') \
                                        .select('clicked', 'entity_id',␣
 ↪'event_country', 'confidence_level_ent','ad_id') \
                                        .groupby('event_country','entity_id').
 ↪agg(F.sum('clicked').alias('clicks'),
                                                                        F.
 ↪count('*').alias('views'),
                                                                        F.
 ↪mean('confidence_level_ent').alias('avg_confidence_level_ent'),
                                                                        F.
 ↪countDistinct('ad_id').alias('distinct_ad_ids')) \
                                         .withColumn('ctr',␣
 ↪ctr_udf('clicks','views'))

entity_id_by_country_popularity = entity_id_by_country_popularity_df.
 ↪filter('views > 5 and event_country <> ""').select('event_country',␣
 ↪'entity_id', 'ctr', 'views', 'avg_confidence_level_ent', 'distinct_ad_ids') \
                .rdd.map(lambda x: ((x['event_country'], x['entity_id']),␣
 ↪(x['ctr'], x['views'], x['distinct_ad_ids'],␣
 ↪x['avg_confidence_level_ent']))).collectAsMap()
len(entity_id_by_country_popularity)
```

[120]:
```
217878
```

[121]:
```
entity_id_by_country_popularity_broad = sc.
 ↪broadcast(entity_id_by_country_popularity)
```

3.2.15 Loading # docs by categories, topics, entities

[122]:
```
#import cPickle
import _pickle as cPickle
```

[123]:
```
df_filenames_suffix = ''
if evaluation:
    df_filenames_suffix = '_eval'
```

[128]:
```
with open('aux_data/categories_docs_counts'+df_filenames_suffix+'.pickle',␣
 ↪'rb') as input_file:
    categories_docs_counts = cPickle.load(input_file)
len(categories_docs_counts)
```

[129]:
```
with open('aux_data/topics_docs_counts'+df_filenames_suffix+'.pickle', 'rb') as␣
↪input_file:
    topics_docs_counts = cPickle.load(input_file)
len(topics_docs_counts)
```

[130]:
```
with open('aux_data/entities_docs_counts'+df_filenames_suffix+'.pickle', 'rb')␣
↪as input_file:
    entities_docs_counts = cPickle.load(input_file)
len(entities_docs_counts)
```

[127]:
```
documents_total = documents_meta_df.count()
documents_total
```

[127]: 2999334

3.3 Exploring Publish Time

[131]:
```
publish_times_df = train_set_df.filter('publish_time is not null').
↪select('document_id_promo','publish_time').distinct().select(F.
↪col('publish_time').cast(IntegerType()))
publish_time_percentiles = get_percentiles(publish_times_df, 'publish_time',␣
↪quantiles_levels=[0.5], max_error_rate=0.001)
publish_time_percentiles
```

[131]: {0.5: 1464112800.0}

[132]:
```
publish_time_median = int(publish_time_percentiles[0.5])
datetime.datetime.utcfromtimestamp(publish_time_median)
```

[132]: datetime.datetime(2016, 5, 24, 18, 0)

[133]:
```
def get_days_diff(newer_timestamp, older_timestamp):
    sec_diff = newer_timestamp - older_timestamp
    days_diff = sec_diff / 60 / 60 / 24
    return days_diff

def get_time_decay_factor(timestamp, timestamp_ref=None, alpha=0.001):
    if timestamp_ref == None:
        timestamp_ref = time.time()

    days_diff = get_days_diff(timestamp_ref, timestamp)
    denominator = math.pow(1+alpha, days_diff)
    if denominator != 0:
        return 1.0 / denominator
    else:
```

```
        return 0.0
```

[134]:
```
def convert_odd_timestamp(timestamp_ms_relative):
    TIMESTAMP_DELTA=1465876799998
    return datetime.datetime.
 ↪fromtimestamp((int(timestamp_ms_relative)+TIMESTAMP_DELTA)//1000)
```

[135]:
```
TIME_DECAY_ALPHA = 0.0005
```

[136]:
```
ref_dates = [
                1476714880, # 7 days
                1474727680, # 30 days
                1469370880, # 90 days
                1461508480,  # 180 days
                1445697280, # 1 year
                1414161280 # 2 years
]

for d in ref_dates:
    print(datetime.datetime.utcfromtimestamp(d), get_time_decay_factor(d,␣
 ↪alpha=TIME_DECAY_ALPHA))
```

```
2016-10-17 14:34:40 0.38367507884058083
2016-09-24 14:34:40 0.37928917906500875
2016-07-24 14:34:40 0.3677144430617202
2016-04-24 14:34:40 0.3513623538857341
2015-10-24 14:34:40 0.3206470243809176
2014-10-24 14:34:40 0.2671703621436017
```

3.3.1 Get local time

[137]:
```
DEFAULT_TZ_EST = -4.0
```

[138]:
```
def get_local_utc_bst_tz(event_country, event_country_state):
    local_tz = DEFAULT_TZ_EST
    if len(event_country) > 0:
        if event_country in countries_utc_dst_broad.value:
            local_tz = countries_utc_dst_broad.value[event_country]
            if len(event_country_state)>2:
                state = event_country_state[3:5]
                if event_country == 'US':
                    if state in us_states_utc_dst_broad.value:
                        local_tz = us_states_utc_dst_broad.value[state]
                elif event_country == 'CA':
                    if state in ca_countries_utc_dst_broad.value:
                        local_tz = ca_countries_utc_dst_broad.value[state]
    return float(local_tz)
```

[139]:
```
hour_bins_dict = {'EARLY_MORNING': 1,
                  'MORNING': 2,
                  'MIDDAY': 3,
                  'AFTERNOON': 4,
                  'EVENING': 5,
                  'NIGHT': 6}

hour_bins_values = sorted(hour_bins_dict.values())
```

[140]:
```
def get_hour_bin(hour):
    if hour >= 5 and hour < 8:
        hour_bin = hour_bins_dict['EARLY_MORNING']
    elif hour >= 8 and hour < 11:
        hour_bin = hour_bins_dict['MORNING']
    elif hour >= 11 and hour < 14:
        hour_bin = hour_bins_dict['MIDDAY']
    elif hour >= 14 and hour < 19:
        hour_bin = hour_bins_dict['AFTERNOON']
    elif hour >= 19 and hour < 22:
        hour_bin = hour_bins_dict['EVENING']
    else:
        hour_bin = hour_bins_dict['NIGHT']
    return hour_bin
```

[141]:
```
def get_local_datetime(dt, event_country, event_country_state):
    local_tz = get_local_utc_bst_tz(event_country, event_country_state)
    tz_delta = local_tz - DEFAULT_TZ_EST
    local_time = dt +  datetime.timedelta(hours=tz_delta)
    return local_time
```

[142]:
```
get_local_datetime(datetime.datetime.now(), 'US', 'US>CA')
```

[142]:
```
datetime.datetime(2022, 1, 15, 21, 7, 8, 294240)
```

[143]:
```
def is_weekend(dt):
    return dt.weekday() >= 5
```

[144]:
```
is_weekend(datetime.datetime(2016, 6, 14))
```

[144]:
```
False
```

3.4 Average CTR functions

[145]:
```
timestamp_ref = date_time_to_unix_epoch(datetime.datetime(2016, 6, 29, 3, 59,␣
↪59))
decay_factor_default = get_time_decay_factor(publish_time_median,␣
↪timestamp_ref, alpha=TIME_DECAY_ALPHA)
```

```
print("decay_factor_default", decay_factor_default)
```

```
decay_factor_default 0.9824518913943062
```

[146]:
```
def get_confidence_sample_size(sample, max_for_reference=100000):
    #Avoiding overflow for large sample size
    if sample >= max_for_reference:
        return 1.0

    ref_log = math.log(1+max_for_reference, 2) #Curiosly reference in log  with␣
 ↪base 2 gives a slightly higher score, so I will keep

    return math.log(1+sample) / float(ref_log)

for i in [0,0.
 ↪5,1,2,3,4,5,10,20,30,100,200,300,1000,2000,3000,10000,20000,30000, 50000,␣
 ↪90000, 100000, 500000, 900000, 1000000, 2171607]:
    print(i, get_confidence_sample_size(i))
```

```
0 0.0
0.5 0.024411410743763327
1 0.041731582304281624
2 0.06614299304804495
3 0.08346316460856325
4 0.09689773339641579
5 0.10787457535232657
10 0.14436755531919657
20 0.183298356035222
30 0.20674645107847822
100 0.2778577004917695
200 0.3192904933647466
300 0.34360197720285013
1000 0.41594812296601125
2000 0.4576496248565576
3000 0.48205100545505175
10000 0.5545232830964639
20000 0.5962518553291584
30000 0.6206622626822822
50000 0.6514162003061013
90000 0.6868039178501281
100000 1.0
500000 1.0
900000 1.0
1000000 1.0
2171607 1.0
```

[147]:
```
def get_popularity(an_id, a_dict):
    return (a_dict[an_id][0], get_confidence_sample_size(a_dict[an_id][1] /␣
↪float(a_dict[an_id][2])) * a_dict[an_id][3]) if an_id in a_dict else (None,␣
↪None)
```

[148]:
```
def get_weighted_avg_popularity_from_list(ids_list, confidence_ids_list,␣
↪pop_dict):
    pops = list(filter(lambda x: x[0][0]!=None, [(get_popularity(an_id,␣
↪pop_dict), confidence) for an_id, confidence in zip(ids_list,␣
↪confidence_ids_list)]))
    #print("pops",pops)
    if len(pops) > 0:
        weighted_avg = sum(map(lambda x: x[0][0]*x[0][1]*x[1], pops)) /␣
↪float(sum(map(lambda x: x[0][1]*x[1], pops)))
        confidence = max(map(lambda x: x[0][1]*x[1], pops))
        return weighted_avg, confidence
    else:
        return None, None
```

[149]:
```
def get_weighted_avg_country_popularity_from_list(event_country, ids_list,␣
↪confidence_ids_list, pop_dict):
    pops = list(filter(lambda x: x[0][0]!=None,␣
↪[(get_popularity((event_country, an_id), pop_dict), confidence) for an_id,␣
↪confidence in zip(ids_list, confidence_ids_list)]))

    if len(pops) > 0:
        weighted_avg = sum(map(lambda x: x[0][0]*x[0][1]*x[1], pops)) /␣
↪float(sum(map(lambda x: x[0][1]*x[1], pops)))
        confidence = max(map(lambda x: x[0][1]*x[1], pops))
        return weighted_avg, confidence
    else:
        return None, None
```

[150]:
```
def get_popularity_score(event_country, ad_id, document_id, source_id,
                         publisher_id, advertiser_id, campaign_id,␣
↪document_id_event,
                            category_ids_by_doc, cat_confidence_level_by_doc,
                            topic_ids_by_doc, top_confidence_level_by_doc,
                            entity_ids_by_doc, ent_confidence_level_by_doc,
                            output_detailed_list=False):
    probs = []

    avg_ctr, confidence = get_popularity(ad_id, ad_id_popularity_broad.value)
    if avg_ctr != None:
        probs.append(('pop_ad_id', avg_ctr, confidence))
```

```
    avg_ctr, confidence = get_popularity(document_id,␣
↪document_id_popularity_broad.value)
    if avg_ctr != None:
        probs.append(('pop_document_id', avg_ctr, confidence))

    avg_ctr, confidence = get_popularity((document_id_event, document_id),␣
↪doc_event_doc_ad_avg_ctr_broad.value)
    if avg_ctr != None:
        probs.append(('pop_doc_event_doc_ad', avg_ctr, confidence))


    if source_id != -1:
        avg_ctr = None
        if event_country != '':
            avg_ctr, confidence = get_popularity((event_country, source_id),␣
↪source_id_by_country_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_source_id_country', avg_ctr, confidence))

        avg_ctr, confidence = get_popularity(source_id,␣
↪source_id_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_source_id', avg_ctr, confidence))


    if publisher_id != None:
        avg_ctr, confidence = get_popularity(publisher_id,␣
↪publisher_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_publisher_id', avg_ctr, confidence))

    if advertiser_id != None:
        avg_ctr, confidence = get_popularity(advertiser_id,␣
↪advertiser_id_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_advertiser_id', avg_ctr, confidence))

    if campaign_id != None:
        avg_ctr, confidence = get_popularity(campaign_id,␣
↪campaign_id_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_campain_id', avg_ctr, confidence))

    if len(entity_ids_by_doc) > 0:
        avg_ctr = None
        if event_country != '':
```

```
            avg_ctr, confidence =␣
↪get_weighted_avg_country_popularity_from_list(event_country,␣
↪entity_ids_by_doc, ent_confidence_level_by_doc,
                                        entity_id_by_country_popularity_broad.
↪value)
        if avg_ctr != None:
            probs.append(('pop_entity_id_country', avg_ctr, confidence))

        avg_ctr, confidence =␣
↪get_weighted_avg_popularity_from_list(entity_ids_by_doc,␣
↪ent_confidence_level_by_doc,
                                                                        ␣
↪entity_id_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_entity_id', avg_ctr, confidence))



    if len(topic_ids_by_doc) > 0:
        avg_ctr = None
        if event_country != '':
            avg_ctr, confidence =␣
↪get_weighted_avg_country_popularity_from_list(event_country,␣
↪topic_ids_by_doc, top_confidence_level_by_doc,
                                        topic_id_id_by_country_popularity_broad.
↪value)
        if avg_ctr != None:
            probs.append(('pop_topic_id_country', avg_ctr, confidence))

        avg_ctr, confidence =␣
↪get_weighted_avg_popularity_from_list(topic_ids_by_doc,␣
↪top_confidence_level_by_doc,
                                                                        ␣
↪topic_id_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_topic_id', avg_ctr, confidence))


    if len(category_ids_by_doc) > 0:
        avg_ctr = None
        if event_country != '':
            avg_ctr, confidence =␣
↪get_weighted_avg_country_popularity_from_list(event_country,␣
↪category_ids_by_doc, cat_confidence_level_by_doc,
                                        category_id_by_country_popularity_broad.
↪value)
```

```
        if avg_ctr != None:
            probs.append(('pop_category_id_country', avg_ctr, confidence))

        avg_ctr, confidence =␣
↪get_weighted_avg_popularity_from_list(category_ids_by_doc,␣
↪cat_confidence_level_by_doc,
                                                                    ␣
↪category_id_popularity_broad.value)
        if avg_ctr != None:
            probs.append(('pop_category_id', avg_ctr, confidence))

    #print("[get_popularity_score] probs", probs)
    if output_detailed_list:
        return probs

    else:
        if len(probs) > 0:
            #weighted_avg_probs_by_confidence = sum(map(lambda x: x[1] *  math.
↪log(1+x[2],2), probs)) / float(sum(map(lambda x: math.log(1+x[2],2),␣
↪probs)))
            weighted_avg_probs_by_confidence = sum(map(lambda x: x[1] * x[2],␣
↪probs)) / float(sum(map(lambda x: x[2], probs)))
            confidence = max(map(lambda x: x[2], probs))
            return weighted_avg_probs_by_confidence, confidence
        else:
            return None, None
```

3.5 Content-Based similarity functions

[151]:
```
def cosine_similarity_dicts(dict1, dict2):
    dict1_norm = math.sqrt(sum([v**2 for v in dict1.values()]))
    dict2_norm = math.sqrt(sum([v**2 for v in dict2.values()]))

    sum_common_aspects = 0.0
    intersections = 0
    for key in dict1:
        if key in dict2:
            sum_common_aspects += dict1[key] * dict2[key]
            intersections += 1

    return sum_common_aspects / (dict1_norm * dict2_norm), intersections
```

[152]:
```
def cosine_similarity_user_docs_aspects(user_aspect_profile, doc_aspect_ids,␣
↪doc_aspects_confidence, aspect_docs_counts):
    if user_aspect_profile==None or len(user_aspect_profile) == 0 or␣
↪doc_aspect_ids == None or len(doc_aspect_ids) == 0:
```

```
        return None, None

    doc_aspects = dict(zip(doc_aspect_ids, doc_aspects_confidence))
    doc_aspects_tfidf_confid = {}
    for key in doc_aspects:
        tf = 1.0
        idf = math.log(math.log(documents_total /␣
↪float(aspect_docs_counts[key])))
        confidence = doc_aspects[key]
        doc_aspects_tfidf_confid[key] = tf*idf * confidence

    user_aspects_tfidf_confid = {}
    for key in user_aspect_profile:
        tfidf = user_aspect_profile[key][0]
        confidence = user_aspect_profile[key][1]
        user_aspects_tfidf_confid[key] = tfidf * confidence

    similarity, intersections =␣
↪cosine_similarity_dicts(doc_aspects_tfidf_confid, user_aspects_tfidf_confid)

    if intersections > 0:
        #P(A intersect B)_intersections = P(A)^intersections *␣
↪P(B)^intersections
        random_error = math.pow(len(doc_aspects)         /␣
↪float(len(aspect_docs_counts)), intersections) * \
                       math.pow(len(user_aspect_profile) /␣
↪float(len(aspect_docs_counts)), intersections)
        confidence = 1.0 - random_error
    else:
        #P(A not intersect B) = 1 - P(A intersect B)
        random_error = 1 - ((len(doc_aspects) / float(len(aspect_docs_counts)))␣
↪* \
                            (len(user_aspect_profile) /␣
↪float(len(aspect_docs_counts))))

    confidence = 1.0 - random_error

    return similarity, confidence
```

[153]:
```
def cosine_similarity_doc_event_doc_ad_aspects(doc_event_aspect_ids,␣
↪doc_event_aspects_confidence,
                                               doc_ad_aspect_ids,␣
↪doc_ad_aspects_confidence,
                                               aspect_docs_counts):
    if doc_event_aspect_ids == None or len(doc_event_aspect_ids) == 0 or \
       doc_ad_aspect_ids == None or len(doc_ad_aspect_ids) == 0:
```

```
        return None, None

    doc_event_aspects = dict(zip(doc_event_aspect_ids,␣
↪doc_event_aspects_confidence))
    doc_event_aspects_tfidf_confid = {}
    for key in doc_event_aspect_ids:
        tf = 1.0
        idf = math.log(math.log(documents_total /␣
↪float(aspect_docs_counts[key])))
        confidence = doc_event_aspects[key]
        doc_event_aspects_tfidf_confid[key] = tf*idf * confidence

    doc_ad_aspects = dict(zip(doc_ad_aspect_ids, doc_ad_aspects_confidence))
    doc_ad_aspects_tfidf_confid = {}
    for key in doc_ad_aspect_ids:
        tf = 1.0
        idf = math.log(math.log(documents_total /␣
↪float(aspect_docs_counts[key])))
        confidence = doc_ad_aspects[key]
        doc_ad_aspects_tfidf_confid[key] = tf*idf * confidence

    similarity, intersections =␣
↪cosine_similarity_dicts(doc_event_aspects_tfidf_confid,␣
↪doc_ad_aspects_tfidf_confid)

    if intersections > 0:
        #P(A intersect B)_intersections = P(A)^intersections *␣
↪P(B)^intersections
        random_error = math.pow(len(doc_event_aspect_ids) /␣
↪float(len(aspect_docs_counts)), intersections) * \
                       math.pow(len(doc_ad_aspect_ids) /␣
↪float(len(aspect_docs_counts)), intersections)
        confidence = 1.0 - random_error
    else:
        #P(A not intersect B) = 1 - P(A intersect B)
        random_error = 1 - ((len(doc_event_aspect_ids) /␣
↪float(len(aspect_docs_counts))) * \
                            (len(doc_ad_aspect_ids) /␣
↪float(len(aspect_docs_counts))))

    confidence = 1.0 - random_error

    return similarity, confidence
```

[154]:
```
def get_user_cb_interest_score(user_views_count, user_categories, user_topics,␣
↪user_entities,
```

```
                            timestamp_event, category_ids_by_doc,␣
↪cat_confidence_level_by_doc,
                            topic_ids_by_doc, top_confidence_level_by_doc,
                            entity_ids_by_doc, ent_confidence_level_by_doc,
                            output_detailed_list=False):

    #Content-Based

    sims = []

    categories_similarity, cat_sim_confidence =␣
↪cosine_similarity_user_docs_aspects(user_categories, category_ids_by_doc,␣
↪cat_confidence_level_by_doc, categories_docs_counts)
    if categories_similarity != None:
        sims.append(('user_doc_ad_sim_categories', categories_similarity,␣
↪cat_sim_confidence))

    topics_similarity, top_sim_confidence =␣
↪cosine_similarity_user_docs_aspects(user_topics, topic_ids_by_doc,␣
↪top_confidence_level_by_doc, topics_docs_counts)
    if topics_similarity != None:
        sims.append(('user_doc_ad_sim_topics', topics_similarity,␣
↪top_sim_confidence))

    entities_similarity, entity_sim_confid =␣
↪cosine_similarity_user_docs_aspects(user_entities, entity_ids_by_doc,␣
↪ent_confidence_level_by_doc, entities_docs_counts)
    if entities_similarity != None:
        sims.append(('user_doc_ad_sim_entities', entities_similarity,␣
↪entity_sim_confid))

    if output_detailed_list:
        return sims
    else:
        if len(sims) > 0:
            weighted_avg_sim_by_confidence = sum(map(lambda x: x[1]*x[2],␣
↪sims)) / float(sum(map(lambda x: x[2], sims)))
            confidence = sum(map(lambda x: x[2], sims)) / float(len(sims))

            #print("[get_user_cb_interest_score] sims: {} | Avg: {} - Confid:␣
↪{}".format(sims, weighted_avg_sim_by_confidence, confidence))
            return weighted_avg_sim_by_confidence, confidence
        else:
            return None, None
```

[155]:
```
def get_doc_event_doc_ad_cb_similarity_score(doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                                             doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                                             doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels,
                                             doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
                                             doc_ad_topic_ids,␣
↪doc_ad_top_confidence_levels,
                                             doc_ad_entity_ids,␣
↪doc_ad_ent_confidence_levels,
                             output_detailed_list=False):

    #Content-Based
    sims = []



    categories_similarity, cat_sim_confidence =␣
↪cosine_similarity_doc_event_doc_ad_aspects(
                                                  doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                                                  doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
                                                  categories_docs_counts)
    if categories_similarity != None:
        sims.append(('doc_event_doc_ad_sim_categories', categories_similarity,␣
↪cat_sim_confidence))

    topics_similarity, top_sim_confidence =␣
↪cosine_similarity_doc_event_doc_ad_aspects(
                                                  doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                                                  doc_ad_topic_ids,␣
↪doc_ad_top_confidence_levels,
                                                  topics_docs_counts)

    if topics_similarity != None:
        sims.append(('doc_event_doc_ad_sim_topics', topics_similarity,␣
↪top_sim_confidence))

    entities_similarity, entity_sim_confid =␣
↪cosine_similarity_doc_event_doc_ad_aspects(
                                                  doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels,
```

```
                                                    doc_ad_entity_ids,␣
↪doc_ad_ent_confidence_levels,
                                                    entities_docs_counts)

    if entities_similarity != None:
        sims.append(('doc_event_doc_ad_sim_entities', entities_similarity,␣
↪entity_sim_confid))

    if output_detailed_list:
        return sims
    else:
        if len(sims) > 0:
            weighted_avg_sim_by_confidence = sum(map(lambda x: x[1]*x[2],␣
↪sims)) / float(sum(map(lambda x: x[2], sims)))
            confidence = sum(map(lambda x: x[2], sims)) / float(len(sims))

            #print("[get_user_cb_interest_score] sims: {} | Avg: {} - Confid:␣
↪{}".format(sims, weighted_avg_sim_by_confidence, confidence))
            return weighted_avg_sim_by_confidence, confidence
        else:
            return None, None
```

4 Feature Vector export

[156]:
```
bool_feature_names = ['event_weekend',
                      'user_has_already_viewed_doc']
```

[157]:
```
int_feature_names = ['user_views',
                     'ad_views',
                     'doc_views',
                     'doc_event_days_since_published',
                     'doc_event_hour',
                     'doc_ad_days_since_published',
                     ]
```

[158]:
```
float_feature_names = [
                'pop_ad_id',
                'pop_ad_id_conf',
                'pop_ad_id_conf_multipl',
                'pop_document_id',
                'pop_document_id_conf',
                'pop_document_id_conf_multipl',
                'pop_publisher_id',
                'pop_publisher_id_conf',
                'pop_publisher_id_conf_multipl',
                'pop_advertiser_id',
```

```
                'pop_advertiser_id_conf',
                'pop_advertiser_id_conf_multipl',
                'pop_campain_id',
                'pop_campain_id_conf',
                'pop_campain_id_conf_multipl',
                'pop_doc_event_doc_ad',
                'pop_doc_event_doc_ad_conf',
                'pop_doc_event_doc_ad_conf_multipl',
                'pop_source_id',
                'pop_source_id_conf',
                'pop_source_id_conf_multipl',
                'pop_source_id_country',
                'pop_source_id_country_conf',
                'pop_source_id_country_conf_multipl',
                'pop_entity_id',
                'pop_entity_id_conf',
                'pop_entity_id_conf_multipl',
                'pop_entity_id_country',
                'pop_entity_id_country_conf',
                'pop_entity_id_country_conf_multipl',
                'pop_topic_id',
                'pop_topic_id_conf',
                'pop_topic_id_conf_multipl',
                'pop_topic_id_country',
                'pop_topic_id_country_conf',
                'pop_topic_id_country_conf_multipl',
                'pop_category_id',
                'pop_category_id_conf',
                'pop_category_id_conf_multipl',
                'pop_category_id_country',
                'pop_category_id_country_conf',
                'pop_category_id_country_conf_multipl',
                'user_doc_ad_sim_categories',
                'user_doc_ad_sim_categories_conf',
                'user_doc_ad_sim_categories_conf_multipl',
                'user_doc_ad_sim_topics',
                'user_doc_ad_sim_topics_conf',
                'user_doc_ad_sim_topics_conf_multipl',
                'user_doc_ad_sim_entities',
                'user_doc_ad_sim_entities_conf',
                'user_doc_ad_sim_entities_conf_multipl',
                'doc_event_doc_ad_sim_categories',
                'doc_event_doc_ad_sim_categories_conf',
                'doc_event_doc_ad_sim_categories_conf_multipl',
                'doc_event_doc_ad_sim_topics',
                'doc_event_doc_ad_sim_topics_conf',
                'doc_event_doc_ad_sim_topics_conf_multipl',
```

```
                'doc_event_doc_ad_sim_entities',
                'doc_event_doc_ad_sim_entities_conf',
                'doc_event_doc_ad_sim_entities_conf_multipl'
               ]
```

[159]:
```
TRAFFIC_SOURCE_FV='traffic_source'
EVENT_HOUR_FV='event_hour'
EVENT_COUNTRY_FV = 'event_country'
EVENT_COUNTRY_STATE_FV = 'event_country_state'
EVENT_GEO_LOCATION_FV = 'event_geo_location'
EVENT_PLATFORM_FV = 'event_platform'
AD_ADVERTISER_FV = 'ad_advertiser'
DOC_AD_SOURCE_ID_FV='doc_ad_source_id'
DOC_AD_PUBLISHER_ID_FV='doc_ad_publisher_id'
DOC_EVENT_SOURCE_ID_FV='doc_event_source_id'
DOC_EVENT_PUBLISHER_ID_FV='doc_event_publisher_id'
DOC_AD_CATEGORY_ID_FV='doc_ad_category_id'
DOC_AD_TOPIC_ID_FV='doc_ad_topic_id'
DOC_AD_ENTITY_ID_FV='doc_ad_entity_id'
DOC_EVENT_CATEGORY_ID_FV='doc_event_category_id'
DOC_EVENT_TOPIC_ID_FV='doc_event_topic_id'
DOC_EVENT_ENTITY_ID_FV='doc_event_entity_id'
```

4.0.1 Configuring feature vector

[160]:
```
category_feature_names_integral = ['ad_advertiser',
 'doc_ad_category_id_1',
 'doc_ad_category_id_2',
 'doc_ad_category_id_3',
 'doc_ad_topic_id_1',
 'doc_ad_topic_id_2',
 'doc_ad_topic_id_3',
 'doc_ad_entity_id_1',
 'doc_ad_entity_id_2',
 'doc_ad_entity_id_3',
 'doc_ad_entity_id_4',
 'doc_ad_entity_id_5',
 'doc_ad_entity_id_6',
 'doc_ad_publisher_id',
 'doc_ad_source_id',
 'doc_event_category_id_1',
 'doc_event_category_id_2',
 'doc_event_category_id_3',
 'doc_event_topic_id_1',
 'doc_event_topic_id_2',
 'doc_event_topic_id_3',
 'doc_event_entity_id_1',
```

```
 'doc_event_entity_id_2',
 'doc_event_entity_id_3',
 'doc_event_entity_id_4',
 'doc_event_entity_id_5',
 'doc_event_entity_id_6',
 'doc_event_publisher_id',
 'doc_event_source_id',
 'event_country',
 'event_country_state',
 'event_geo_location',
 'event_hour',
 'event_platform',
 'traffic_source']


feature_vector_labels_integral = bool_feature_names + int_feature_names +␣
↪float_feature_names + \
                                 category_feature_names_integral
```

[161]:
```
feature_vector_labels_integral_dict = dict([(key, idx) for idx, key in␣
↪enumerate(feature_vector_labels_integral)])
```

[162]:
```
with open('feature_vector_labels_integral.txt', 'w') as output:
    output.writelines('\n'.join(feature_vector_labels_integral))
```

[163]:
```
def set_feature_vector_cat_value(field_name, field_value, feature_vector):
    if not is_null(field_value) and str(field_value) != '-1':
        feature_name = get_ohe_feature_name(field_name, field_value)
        if feature_name in feature_vector_labels_dict:
            feature_idx = feature_vector_labels_dict[feature_name]
        else:
            #Unpopular category value
            feature_idx =␣
↪feature_vector_labels_dict[get_ohe_feature_name(field_name,␣
↪LESS_SPECIAL_CAT_VALUE)]

        feature_vector[feature_idx] = float(1)

def set_feature_vector_cat_values(field_name, field_values, feature_vector):
    for field_value in field_values:
        set_feature_vector_cat_value(field_name, field_value, feature_vector)
```

[164]:
```
def get_ad_feature_vector(user_doc_ids_viewed, user_views_count,␣
↪user_categories, user_topics, user_entities,
                          event_country, event_country_state,
                          ad_id, document_id, source_id, doc_ad_publish_time,␣
↪timestamp_event, platform_event,
```

```
                            geo_location_event,
                            doc_event_source_id, doc_event_publisher_id,␣
↪doc_event_publish_time,
                            traffic_source_pv, advertiser_id, publisher_id,
                            campaign_id, document_id_event,
                            doc_ad_category_ids, doc_ad_cat_confidence_levels,
                            doc_ad_topic_ids, doc_ad_top_confidence_levels,
                            doc_ad_entity_ids, doc_ad_ent_confidence_levels,
                            doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                            doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                            doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels):

    try:

        feature_vector = {}

        if user_views_count != None:
            feature_vector[feature_vector_labels_dict['user_views']] =␣
↪float(user_views_count)

        if user_doc_ids_viewed != None:
            ␣
↪feature_vector[feature_vector_labels_dict['user_has_already_viewed_doc']] =␣
↪float(document_id in user_doc_ids_viewed)

        if ad_id in ad_id_popularity_broad.value:
            feature_vector[feature_vector_labels_dict['ad_views']] =␣
↪float(ad_id_popularity_broad.value[ad_id][1])

        if document_id in document_id_popularity_broad.value:
            feature_vector[feature_vector_labels_dict['doc_views']] =␣
↪float(document_id_popularity_broad.value[document_id][1])

        if timestamp_event > -1:
            dt_timestamp_event = convert_odd_timestamp(timestamp_event)
            if doc_ad_publish_time != None:
                delta_days = (dt_timestamp_event - doc_ad_publish_time).days
                if delta_days >= 0 and delta_days <= 365*10: #10 years
                    ␣
↪feature_vector[feature_vector_labels_dict['doc_ad_days_since_published']] =␣
↪float(delta_days)

            if doc_event_publish_time != None:
```

```
                delta_days = (dt_timestamp_event - doc_event_publish_time).days
                if delta_days >= 0 and delta_days <= 365*10: #10 years
                    ␣
↪feature_vector[feature_vector_labels_dict['doc_event_days_since_published']]␣
↪= float(delta_days)


            #Local period of the day (hours)
            dt_local_timestamp_event = get_local_datetime(dt_timestamp_event,␣
↪event_country, event_country_state)
            local_hour_bin = get_hour_bin(dt_local_timestamp_event.hour)
            feature_vector[feature_vector_labels_dict['doc_event_hour']] =␣
↪float(local_hour_bin) #Hour for Decision Trees
            set_feature_vector_cat_value(EVENT_HOUR_FV, local_hour_bin,␣
↪feature_vector) #Period of day for FFM

            #Weekend
            weekend = int(is_weekend(dt_local_timestamp_event))
            feature_vector[feature_vector_labels_dict['event_weekend']] =␣
↪float(weekend)

        conf_field_suffix = '_conf'
        conf_multiplied_field_suffix = '_conf_multipl'

        #Setting Popularity fields
        pop_scores = get_popularity_score(event_country, ad_id, document_id,␣
↪source_id,
                                  publisher_id, advertiser_id, campaign_id,␣
↪document_id_event,
                                  doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
                                  doc_ad_topic_ids, doc_ad_top_confidence_levels,
                                  doc_ad_entity_ids, doc_ad_ent_confidence_levels,
                                  output_detailed_list=True)



        for score in pop_scores:
            feature_vector[feature_vector_labels_dict[score[0]]] = score[1]
            ␣
↪feature_vector[feature_vector_labels_dict[score[0]+conf_field_suffix]] =␣
↪score[2]
            ␣
↪feature_vector[feature_vector_labels_dict[score[0]+conf_multiplied_field_suffix]]␣
↪= score[1] * score[2]
```

```
        #Setting User-Doc_ad CB Similarity fields
        user_doc_ad_cb_sim_scores =␣
↪get_user_cb_interest_score(user_views_count, user_categories, user_topics,␣
↪user_entities,
                                timestamp_event,
                                 doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
                                 doc_ad_topic_ids, doc_ad_top_confidence_levels,
                                 doc_ad_entity_ids,␣
↪doc_ad_ent_confidence_levels,
                                output_detailed_list=True)

        for score in user_doc_ad_cb_sim_scores:
            feature_vector[feature_vector_labels_dict[score[0]]] = score[1]
            ␣
↪feature_vector[feature_vector_labels_dict[score[0]+conf_field_suffix]] =␣
↪score[2]
            ␣
↪feature_vector[feature_vector_labels_dict[score[0]+conf_multiplied_field_suffix]]␣
↪= score[1] * score[2]

        #Setting Doc_event-doc_ad CB Similarity fields
        doc_event_doc_ad_cb_sim_scores =␣
↪get_doc_event_doc_ad_cb_similarity_score(
                                            doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                                            doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                                            doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels,
                                            doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
                                            doc_ad_topic_ids,␣
↪doc_ad_top_confidence_levels,
                                            doc_ad_entity_ids,␣
↪doc_ad_ent_confidence_levels,
                                        output_detailed_list=True)

        for score in doc_event_doc_ad_cb_sim_scores:
            feature_vector[feature_vector_labels_dict[score[0]]] = score[1]
            ␣
↪feature_vector[feature_vector_labels_dict[score[0]+conf_field_suffix]] =␣
↪score[2]
            ␣
↪feature_vector[feature_vector_labels_dict[score[0]+conf_multiplied_field_suffix]]␣
↪= score[1] * score[2]
```

```
        set_feature_vector_cat_value(TRAFFIC_SOURCE_FV, traffic_source_pv,␣
↪feature_vector)
        set_feature_vector_cat_value(EVENT_COUNTRY_FV, event_country,␣
↪feature_vector)
        set_feature_vector_cat_value(EVENT_COUNTRY_STATE_FV,␣
↪event_country_state, feature_vector)
        set_feature_vector_cat_value(EVENT_GEO_LOCATION_FV, geo_location_event,␣
↪feature_vector)
        set_feature_vector_cat_value(EVENT_PLATFORM_FV, platform_event,␣
↪feature_vector)
        set_feature_vector_cat_value(AD_ADVERTISER_FV, advertiser_id,␣
↪feature_vector)
        set_feature_vector_cat_value(DOC_AD_SOURCE_ID_FV, source_id,␣
↪feature_vector)
        set_feature_vector_cat_value(DOC_AD_PUBLISHER_ID_FV, publisher_id,␣
↪feature_vector)
        set_feature_vector_cat_value(DOC_EVENT_SOURCE_ID_FV,␣
↪doc_event_source_id, feature_vector)
        set_feature_vector_cat_value(DOC_EVENT_PUBLISHER_ID_FV,␣
↪doc_event_publisher_id, feature_vector)
        set_feature_vector_cat_values(DOC_AD_CATEGORY_ID_FV,␣
↪doc_ad_category_ids, feature_vector)
        set_feature_vector_cat_values(DOC_AD_TOPIC_ID_FV, doc_ad_topic_ids,␣
↪feature_vector)
        set_feature_vector_cat_values(DOC_AD_ENTITY_ID_FV, doc_ad_entity_ids,␣
↪feature_vector)
        set_feature_vector_cat_values(DOC_EVENT_CATEGORY_ID_FV,␣
↪doc_event_category_ids, feature_vector)
        set_feature_vector_cat_values(DOC_EVENT_TOPIC_ID_FV,␣
↪doc_event_topic_ids, feature_vector)
        set_feature_vector_cat_values(DOC_EVENT_ENTITY_ID_FV,␣
↪doc_event_entity_ids, feature_vector)

        #Creating dummy column as the last column because xgboost have a␣
↪problem if the last column is undefined for all rows,
        #saying that dimentions of data and feature_names do not match
        #feature_vector[feature_vector_labels_dict[DUMMY_FEATURE_COLUMN]] =␣
↪float(0)

        #Ensuring that all elements are floats for compatibility with UDF␣
↪output (ArrayType(FloatType()))
        #feature_vector = list([float(x) for x in feature_vector])

    except Exception as e:
```

```
        raise Exception("[get_ad_feature_vector] ERROR PROCESSING FEATURE␣
↪VECTOR! Params: {}" \
                        .format([user_doc_ids_viewed, user_views_count,␣
↪user_categories, user_topics, user_entities,
                                 event_country, event_country_state,
                             ad_id, document_id, source_id, doc_ad_publish_time,␣
↪timestamp_event, platform_event,
                             geo_location_event,
                             doc_event_source_id, doc_event_publisher_id,␣
↪doc_event_publish_time,
                             traffic_source_pv, advertiser_id, publisher_id,
                             campaign_id, document_id_event,
                             doc_ad_category_ids, doc_ad_cat_confidence_levels,
                             doc_ad_topic_ids, doc_ad_top_confidence_levels,
                             doc_ad_entity_ids, doc_ad_ent_confidence_levels,
                             doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                             doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                             doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels]),
                        e)

    return SparseVector(len(feature_vector_labels_dict), feature_vector)
```

[165]:
```
get_ad_feature_vector_udf = F.udf(lambda user_doc_ids_viewed, user_views_count,␣
↪user_categories, user_topics,
                                        user_entities, event_country,␣
↪event_country_state, ad_id, document_id, source_id,
                                        doc_ad_publish_time, timestamp_event,␣
↪platform_event,
                                        geo_location_event,
                                        doc_event_source_id,␣
↪doc_event_publisher_id, doc_event_publish_time,
                                        traffic_source_pv, advertiser_id,␣
↪publisher_id,
                                        campaign_id, document_id_event,
                                        category_ids_by_doc,␣
↪cat_confidence_level_by_doc,
                                        topic_ids_by_doc,␣
↪top_confidence_level_by_doc,
                                        entity_ids_by_doc,␣
↪ent_confidence_level_by_doc,
                                        doc_event_category_id_list,␣
↪doc_event_confidence_level_cat_list,
```

```
                                        doc_event_topic_id_list,␣
↪doc_event_confidence_level_top,
                                        doc_event_entity_id_list,␣
↪doc_event_confidence_level_ent: \
                                        ␣
↪get_ad_feature_vector(user_doc_ids_viewed, user_views_count,␣
↪user_categories, user_topics, user_entities,
                                                            event_country,␣
↪event_country_state,
                                                            ad_id, document_id,␣
↪source_id, doc_ad_publish_time, timestamp_event, platform_event,
                                                            geo_location_event,
                                                           ␣
↪doc_event_source_id, doc_event_publisher_id, doc_event_publish_time,
                                                            traffic_source_pv,␣
↪advertiser_id, publisher_id,
                                                            campaign_id,␣
↪document_id_event,
                                                           ␣
↪category_ids_by_doc, cat_confidence_level_by_doc,
                                                            topic_ids_by_doc,␣
↪top_confidence_level_by_doc,
                                                            entity_ids_by_doc,␣
↪ent_confidence_level_by_doc,
                                                           ␣
↪doc_event_category_id_list, doc_event_confidence_level_cat_list,
                                                           ␣
↪doc_event_topic_id_list, doc_event_confidence_level_top,
                                                           ␣
↪doc_event_entity_id_list, doc_event_confidence_level_ent),
                            VectorUDT())
```

4.0.2 Building feature vectors

[166]:
```
def set_feature_vector_cat_value_integral(field_name, field_value,␣
↪feature_vector):
    if not is_null(field_value): #and str(field_value) != '-1':
        feature_vector[feature_vector_labels_integral_dict[field_name]] =␣
↪float(field_value)

def set_feature_vector_cat_top_multi_values_integral(field_name, values,␣
↪confidences, feature_vector, top=5):
    top_values = list(filter(lambda z: z != -1, map(lambda y: y[0],␣
↪sorted(zip(values, confidences), key=lambda x: -x[1]))))[:top]
    for idx, field_value in list(enumerate(top_values)):
```

```
        set_feature_vector_cat_value_integral('{}_{}'.format(field_name,␣
↪idx+1), field_value, feature_vector)
```

[167]:
```
def get_ad_feature_vector_integral(user_doc_ids_viewed, user_views_count,␣
↪user_categories, user_topics, user_entities,
                            event_country, event_country_state,
                            ad_id, document_id, source_id, doc_ad_publish_time,␣
↪timestamp_event, platform_event,
                            geo_location_event,
                            doc_event_source_id, doc_event_publisher_id,␣
↪doc_event_publish_time,
                            traffic_source_pv, advertiser_id, publisher_id,
                            campaign_id, document_id_event,
                            doc_ad_category_ids, doc_ad_cat_confidence_levels,
                            doc_ad_topic_ids, doc_ad_top_confidence_levels,
                            doc_ad_entity_ids, doc_ad_ent_confidence_levels,
                            doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                            doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                            doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels):

    try:

        feature_vector = {}

        if user_views_count != None:
            feature_vector[feature_vector_labels_integral_dict['user_views']] =␣
↪float(user_views_count)

        if user_doc_ids_viewed != None:
            ␣
↪feature_vector[feature_vector_labels_integral_dict['user_has_already_viewed_doc']]␣
↪= float(document_id in user_doc_ids_viewed)

        if ad_id in ad_id_popularity_broad.value:
            feature_vector[feature_vector_labels_integral_dict['ad_views']] =␣
↪float(ad_id_popularity_broad.value[ad_id][1])

        if document_id in document_id_popularity_broad.value:
            feature_vector[feature_vector_labels_integral_dict['doc_views']] =␣
↪float(document_id_popularity_broad.value[document_id][1])

        if timestamp_event > -1:
            dt_timestamp_event = convert_odd_timestamp(timestamp_event)
```

```
            if doc_ad_publish_time != None:
                delta_days = (dt_timestamp_event - doc_ad_publish_time).days
                if delta_days >= 0 and delta_days <= 365*10: #10 years
                    ␣
↪feature_vector[feature_vector_labels_integral_dict['doc_ad_days_since_published']]␣
↪= float(delta_days)

            if doc_event_publish_time != None:
                delta_days = (dt_timestamp_event - doc_event_publish_time).days
                if delta_days >= 0 and delta_days <= 365*10: #10 years
                    ␣
↪feature_vector[feature_vector_labels_integral_dict['doc_event_days_since_published']]␣
↪= float(delta_days)


            #Local period of the day (hours)
            dt_local_timestamp_event = get_local_datetime(dt_timestamp_event,␣
↪event_country, event_country_state)
            local_hour_bin = get_hour_bin(dt_local_timestamp_event.hour)
           ␣
↪feature_vector[feature_vector_labels_integral_dict['doc_event_hour']] =␣
↪float(local_hour_bin) #Hour for Decision Trees
            set_feature_vector_cat_value_integral(EVENT_HOUR_FV,␣
↪local_hour_bin, feature_vector) #Period of day for FFM

            #Weekend
            weekend = int(is_weekend(dt_local_timestamp_event))
           ␣
↪feature_vector[feature_vector_labels_integral_dict['event_weekend']] =␣
↪float(weekend)


        conf_field_suffix = '_conf'
        conf_multiplied_field_suffix = '_conf_multipl'

        #Setting Popularity fields
        pop_scores = get_popularity_score(event_country, ad_id, document_id,␣
↪source_id,
                                  publisher_id, advertiser_id, campaign_id,␣
↪document_id_event,
                                  doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
                                  doc_ad_topic_ids, doc_ad_top_confidence_levels,
                                  doc_ad_entity_ids, doc_ad_ent_confidence_levels,
                                  output_detailed_list=True)
```

```
        for score in pop_scores:
            feature_vector[feature_vector_labels_integral_dict[score[0]]] =␣
↪score[1]
            ␣
↪feature_vector[feature_vector_labels_integral_dict[score[0]+conf_field_suffix]]␣
↪= score[2]
            ␣
↪feature_vector[feature_vector_labels_integral_dict[score[0]+conf_multiplied_field_suffix]]␣
↪= score[1] * score[2]

        #Setting User-Doc_ad CB Similarity fields
        user_doc_ad_cb_sim_scores =␣
↪get_user_cb_interest_score(user_views_count, user_categories, user_topics,␣
↪user_entities,
                                timestamp_event,
                                 doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
                                 doc_ad_topic_ids, doc_ad_top_confidence_levels,
                                 doc_ad_entity_ids,␣
↪doc_ad_ent_confidence_levels,
                                output_detailed_list=True)

        for score in user_doc_ad_cb_sim_scores:
            feature_vector[feature_vector_labels_integral_dict[score[0]]] =␣
↪score[1]
            ␣
↪feature_vector[feature_vector_labels_integral_dict[score[0]+conf_field_suffix]]␣
↪= score[2]
            ␣
↪feature_vector[feature_vector_labels_integral_dict[score[0]+conf_multiplied_field_suffix]]␣
↪= score[1] * score[2]

        #Setting Doc_event-doc_ad CB Similarity fields
        doc_event_doc_ad_cb_sim_scores =␣
↪get_doc_event_doc_ad_cb_similarity_score(
                                            doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                                            doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                                            doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels,
                                            doc_ad_category_ids,␣
↪doc_ad_cat_confidence_levels,
```

```
                                            doc_ad_topic_ids,␣
↪doc_ad_top_confidence_levels,
                                            doc_ad_entity_ids,␣
↪doc_ad_ent_confidence_levels,
                                        output_detailed_list=True)

        for score in doc_event_doc_ad_cb_sim_scores:
            feature_vector[feature_vector_labels_integral_dict[score[0]]] =␣
↪score[1]
            ␣
↪feature_vector[feature_vector_labels_integral_dict[score[0]+conf_field_suffix]]␣
↪= score[2]
            ␣
↪feature_vector[feature_vector_labels_integral_dict[score[0]+conf_multiplied_field_suffix]]␣
↪= score[1] * score[2]

        #Process code for event_country
        if event_country in event_country_values_counts:
            event_country_code = event_country_values_counts[event_country]
        else:
            event_country_code =␣
↪event_country_values_counts[LESS_SPECIAL_CAT_VALUE]
        set_feature_vector_cat_value_integral(EVENT_COUNTRY_FV,␣
↪event_country_code, feature_vector)

        #Process code for event_country_state
        if event_country_state in event_country_state_values_counts:
            event_country_state_code =␣
↪event_country_state_values_counts[event_country_state]
        else:
            event_country_state_code =␣
↪event_country_state_values_counts[LESS_SPECIAL_CAT_VALUE]
        set_feature_vector_cat_value_integral(EVENT_COUNTRY_STATE_FV,␣
↪event_country_state_code, feature_vector)

        #Process code for geo_location_event
        if geo_location_event in event_geo_location_values_counts:
            geo_location_event_code =␣
↪event_geo_location_values_counts[geo_location_event]
        else:
            geo_location_event_code =␣
↪event_geo_location_values_counts[LESS_SPECIAL_CAT_VALUE]
        set_feature_vector_cat_value_integral(EVENT_GEO_LOCATION_FV,␣
↪geo_location_event_code, feature_vector)
```

```
        set_feature_vector_cat_value_integral(TRAFFIC_SOURCE_FV,␣
↪traffic_source_pv, feature_vector)
        set_feature_vector_cat_value_integral(EVENT_PLATFORM_FV,␣
↪platform_event, feature_vector)
        set_feature_vector_cat_value_integral(AD_ADVERTISER_FV, advertiser_id,␣
↪feature_vector)
        set_feature_vector_cat_value_integral(DOC_AD_SOURCE_ID_FV, source_id,␣
↪feature_vector)
        set_feature_vector_cat_value_integral(DOC_AD_PUBLISHER_ID_FV,␣
↪publisher_id, feature_vector)
        set_feature_vector_cat_value_integral(DOC_EVENT_SOURCE_ID_FV,␣
↪doc_event_source_id, feature_vector)
        set_feature_vector_cat_value_integral(DOC_EVENT_PUBLISHER_ID_FV,␣
↪doc_event_publisher_id, feature_vector)

        set_feature_vector_cat_top_multi_values_integral(DOC_AD_CATEGORY_ID_FV,␣
↪doc_ad_category_ids, doc_ad_cat_confidence_levels, feature_vector, top=3)
        set_feature_vector_cat_top_multi_values_integral(DOC_AD_TOPIC_ID_FV,␣
↪doc_ad_topic_ids, doc_ad_top_confidence_levels, feature_vector, top=3)

      ␣
↪set_feature_vector_cat_top_multi_values_integral(DOC_EVENT_CATEGORY_ID_FV,␣
↪doc_event_category_ids, doc_event_cat_confidence_levels, feature_vector,␣
↪top=3)
        set_feature_vector_cat_top_multi_values_integral(DOC_EVENT_TOPIC_ID_FV,␣
↪doc_event_topic_ids, doc_event_top_confidence_levels, feature_vector, top=3)

        #Process codes for doc_ad_entity_ids
        doc_ad_entity_ids_codes = [doc_entity_id_values_counts[x] if x in␣
↪doc_entity_id_values_counts
                                   else␣
↪doc_entity_id_values_counts[LESS_SPECIAL_CAT_VALUE]
                                   for x in doc_ad_entity_ids]
        set_feature_vector_cat_top_multi_values_integral(DOC_AD_ENTITY_ID_FV,␣
↪doc_ad_entity_ids_codes, doc_ad_ent_confidence_levels, feature_vector, top=6)


        #Process codes for doc_event_entity_ids
        doc_event_entity_ids_codes = [doc_entity_id_values_counts[x] if x in␣
↪doc_entity_id_values_counts
                                   else␣
↪doc_entity_id_values_counts[LESS_SPECIAL_CAT_VALUE]
                                   for x in doc_event_entity_ids]
```

```
      ␣
↪set_feature_vector_cat_top_multi_values_integral(DOC_EVENT_ENTITY_ID_FV,␣
↪doc_event_entity_ids_codes, doc_event_ent_confidence_levels, feature_vector,␣
↪top=6)

       #Creating dummy column as the last column because xgboost have a␣
↪problem if the last column is undefined for all rows,
       #saying that dimentions of data and feature_names do not match
       #feature_vector[feature_vector_labels_dict[DUMMY_FEATURE_COLUMN]] =␣
↪float(0)

       #Ensuring that all elements are floats for compatibility with UDF␣
↪output (ArrayType(FloatType()))
       #feature_vector = list([float(x) for x in feature_vector])

   except Exception as e:
       raise Exception("[get_ad_feature_vector_integral] ERROR PROCESSING␣
↪FEATURE VECTOR! Params: {}" \
                       .format([user_doc_ids_viewed, user_views_count,␣
↪user_categories, user_topics, user_entities,
                                event_country, event_country_state,
                           ad_id, document_id, source_id, doc_ad_publish_time,␣
↪timestamp_event, platform_event,
                           geo_location_event,
                           doc_event_source_id, doc_event_publisher_id,␣
↪doc_event_publish_time,
                           traffic_source_pv, advertiser_id, publisher_id,
                           campaign_id, document_id_event,
                           doc_ad_category_ids, doc_ad_cat_confidence_levels,
                           doc_ad_topic_ids, doc_ad_top_confidence_levels,
                           doc_ad_entity_ids, doc_ad_ent_confidence_levels,
                           doc_event_category_ids,␣
↪doc_event_cat_confidence_levels,
                           doc_event_topic_ids,␣
↪doc_event_top_confidence_levels,
                           doc_event_entity_ids,␣
↪doc_event_ent_confidence_levels]),
                       e)

   return SparseVector(len(feature_vector_labels_integral_dict),␣
↪feature_vector)
```

[168]:
```
get_ad_feature_vector_integral_udf = F.udf(lambda user_doc_ids_viewed,␣
↪user_views_count, user_categories, user_topics,
                                       user_entities, event_country,␣
↪event_country_state, ad_id, document_id, source_id,
```

```
                                        doc_ad_publish_time, timestamp_event,␣
↪platform_event,
                                        geo_location_event,
                                        doc_event_source_id,␣
↪doc_event_publisher_id, doc_event_publish_time,
                                        traffic_source_pv, advertiser_id,␣
↪publisher_id,
                                        campaign_id, document_id_event,
                                        category_ids_by_doc,␣
↪cat_confidence_level_by_doc,
                                        topic_ids_by_doc,␣
↪top_confidence_level_by_doc,
                                        entity_ids_by_doc,␣
↪ent_confidence_level_by_doc,
                                        doc_event_category_id_list,␣
↪doc_event_confidence_level_cat_list,
                                        doc_event_topic_id_list,␣
↪doc_event_confidence_level_top,
                                        doc_event_entity_id_list,␣
↪doc_event_confidence_level_ent: \
                                        ␣
↪get_ad_feature_vector_integral(user_doc_ids_viewed, user_views_count,␣
↪user_categories, user_topics, user_entities,
                                                                event_country,␣
↪event_country_state,
                                                                ad_id, document_id,␣
↪source_id, doc_ad_publish_time, timestamp_event, platform_event,
                                                                geo_location_event,
                                                               ␣
↪doc_event_source_id, doc_event_publisher_id, doc_event_publish_time,
                                                                traffic_source_pv,␣
↪advertiser_id, publisher_id,
                                                                campaign_id,␣
↪document_id_event,
                                                               ␣
↪category_ids_by_doc, cat_confidence_level_by_doc,
                                                                topic_ids_by_doc,␣
↪top_confidence_level_by_doc,
                                                                entity_ids_by_doc,␣
↪ent_confidence_level_by_doc,
                                                               ␣
↪doc_event_category_id_list, doc_event_confidence_level_cat_list,
                                                               ␣
↪doc_event_topic_id_list, doc_event_confidence_level_top,
```

```
↪doc_event_entity_id_list, doc_event_confidence_level_ent),
                            VectorUDT())
                             #StructField("features", VectorUDT()))
                             #MapType(IntegerType(), FloatType()))
```

[7]:
```
from pyspark.ml.recommendation import ALS, ALSModel

def fit_als_model(rank=100, maxIter=20, regParam=0.001, alpha=500.0):
    als = ALS(implicitPrefs=True, seed=seed,
    rank=rank, maxIter=maxIter, regParam=regParam, alpha=alpha,
    numUserBlocks=10, numItemBlocks=10, checkpointInterval=10,
    userCol="uuid_index", itemCol="doc_id", ratingCol="views_log")
    als_model = als.fit(user_item_implicit_log_df)
    return als_model
```

[11]:
```
print ("the Map Score is ", fit_als_model())
```

```
the Map Score is  0.59116
```

4.1 Export Train set feature vectors

[169]:
```
train_set_enriched_df = train_set_df \
                            .join(documents_categories_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_categories_grouped.
↪document_id_cat"), how='left') \
                            .join(documents_topics_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_topics_grouped.
↪document_id_top"), how='left') \
                            .join(documents_entities_grouped_df, on=F.
↪col("document_id_promo") == F.col("documents_entities_grouped.
↪document_id_ent"), how='left') \
                            .join(documents_categories_grouped_df \
                                    .withColumnRenamed('category_id_list',␣
↪'doc_event_category_id_list')
                                    .
↪withColumnRenamed('confidence_level_cat_list',␣
↪'doc_event_confidence_level_cat_list') \
                                    .
↪alias('documents_event_categories_grouped'),
                                on=F.col("document_id_event") == F.
↪col("documents_event_categories_grouped.document_id_cat"),
                                how='left') \
                            .join(documents_topics_grouped_df \
                                    .withColumnRenamed('topic_id_list',␣
↪'doc_event_topic_id_list')
```

```
                                        .
↪withColumnRenamed('confidence_level_top_list',␣
↪'doc_event_confidence_level_top_list') \
                                            .alias('documents_event_topics_grouped'),
                                    on=F.col("document_id_event") == F.
↪col("documents_event_topics_grouped.document_id_top"),
                                    how='left') \
                                .join(documents_entities_grouped_df \
                                            .withColumnRenamed('entity_id_list',␣
↪'doc_event_entity_id_list')
                                        .
↪withColumnRenamed('confidence_level_ent_list',␣
↪'doc_event_confidence_level_ent_list') \
                                        .
↪alias('documents_event_entities_grouped'),
                                    on=F.col("document_id_event") == F.
↪col("documents_event_entities_grouped.document_id_ent"),
                                    how='left') \
                                .
↪select('display_id','uuid_event','event_country','event_country_state','platform_event',
                                     'source_id_doc_event',␣
↪'publisher_doc_event','publish_time_doc_event',
                                            'publish_time',␣
↪'ad_id','document_id_promo','clicked',
                                           'geo_location_event',␣
↪'advertiser_id', 'publisher_id',
                                            'campaign_id', 'document_id_event',
                                            'traffic_source_pv',
                                        ␣
↪int_list_null_to_empty_list_udf('doc_event_category_id_list').
↪alias('doc_event_category_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('doc_event_confidence_level_cat_list').
↪alias('doc_event_confidence_level_cat_list'),
                                        ␣
↪int_list_null_to_empty_list_udf('doc_event_topic_id_list').
↪alias('doc_event_topic_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('doc_event_confidence_level_top_list').
↪alias('doc_event_confidence_level_top_list'),
                                        ␣
↪str_list_null_to_empty_list_udf('doc_event_entity_id_list').
↪alias('doc_event_entity_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('doc_event_confidence_level_ent_list').
↪alias('doc_event_confidence_level_ent_list'),
```

```
                                        int_null_to_minus_one_udf('source_id').
↪alias('source_id'),
                                       ␣
↪int_null_to_minus_one_udf('timestamp_event').alias('timestamp_event'),
                                       ␣
↪int_list_null_to_empty_list_udf('category_id_list').
↪alias('category_id_list'),
                                       ␣
↪float_list_null_to_empty_list_udf('confidence_level_cat_list').
↪alias('confidence_level_cat_list'),
                                       ␣
↪int_list_null_to_empty_list_udf('topic_id_list').alias('topic_id_list'),
                                       ␣
↪float_list_null_to_empty_list_udf('confidence_level_top_list').
↪alias('confidence_level_top_list'),
                                       ␣
↪str_list_null_to_empty_list_udf('entity_id_list').alias('entity_id_list'),
                                       ␣
↪float_list_null_to_empty_list_udf('confidence_level_ent_list').
↪alias('confidence_level_ent_list')
                                       ) \
                            .join(user_profiles_df, on=[F.col("user_profiles.
↪uuid") == F.col("uuid_event")], how='left') \
                            .withColumnRenamed('categories', 'user_categories')␣
↪\
                            .withColumnRenamed('topics', 'user_topics') \
                            .withColumnRenamed('entities', 'user_entities') \
                            .withColumnRenamed('doc_ids',␣
↪'user_doc_ids_viewed') \
                            .withColumnRenamed('views', 'user_views_count')
```

[170]:
```
train_set_feature_vectors_df = train_set_enriched_df \
                                .withColumn('feature_vector',
                                            #get_ad_feature_vector_udf(
                                            get_ad_feature_vector_integral_udf(
                                                                       ␣
↪'user_doc_ids_viewed',
                                                                       ␣
↪'user_views_count',
                                                                       ␣
↪'user_categories',
                                                                        'user_topics',
                                                                        'user_entities',
                                                                        'event_country',
                                                                       ␣
↪'event_country_state',
```

```
                                                    'ad_id',
                                                   ␣
↪'document_id_promo',
                                                    'source_id',
                                                    'publish_time',
                                                   ␣
↪'timestamp_event',
                                                   ␣
↪'platform_event',
                                                   ␣
↪'geo_location_event',
                                                   ␣
↪'source_id_doc_event',
                                                   ␣
↪'publisher_doc_event',
                                                   ␣
↪'publish_time_doc_event',
                                                   ␣
↪'traffic_source_pv',
                                                    'advertiser_id',
                                                    'publisher_id',
                                                    'campaign_id',
                                                   ␣
↪'document_id_event',
                                                   ␣
↪'category_id_list',
                                                   ␣
↪'confidence_level_cat_list',
                                                    'topic_id_list',
                                                   ␣
↪'confidence_level_top_list',
                                                   ␣
↪'entity_id_list',
                                                   ␣
↪'confidence_level_ent_list',
                                                   ␣
↪'doc_event_category_id_list',
                                                   ␣
↪'doc_event_confidence_level_cat_list',
                                                   ␣
↪'doc_event_topic_id_list',
                                                   ␣
↪'doc_event_confidence_level_top_list',
                                                   ␣
↪'doc_event_entity_id_list',
```

```
                                                                    ␣
↪'doc_event_confidence_level_ent_list')) \
                            .select(F.col('uuid_event').alias('uuid'),
                                    'display_id',
                                    'ad_id',
                                    'document_id_event',
                                    F.col('document_id_promo').
↪alias('document_id'),
                                    F.col('clicked').alias('label'),
                                    'feature_vector') #\
                            #.orderBy('display_id','ad_id')
```

[171]:
```
if evaluation:
    train_feature_vector_gcs_folder_name = 'train_feature_vectors_integral_eval'
else:
    train_feature_vector_gcs_folder_name = 'train_feature_vectors_integral'
```

[173]:
```
%time train_set_feature_vectors_df.write.
↪parquet(OUTPUT_BUCKET_FOLDER+train_feature_vector_gcs_folder_name,␣
↪mode='overwrite')
```

4.2 Exporting integral feature vectors to CSV

[175]:
```
train_feature_vectors_exported_df = spark.read.
↪parquet(OUTPUT_BUCKET_FOLDER+train_feature_vector_gcs_folder_name)
train_feature_vectors_exported_df.take(3)
```

[176]:
```
if evaluation:
    train_feature_vector_integral_csv_folder_name =␣
↪'train_feature_vectors_integral_eval.csv'
else:
    train_feature_vector_integral_csv_folder_name =␣
↪'train_feature_vectors_integral.csv'
```

[177]:
```
integral_headers = ['label', 'display_id', 'ad_id', 'doc_id', 'doc_event_id',␣
↪'is_leak'] + feature_vector_labels_integral

with open(train_feature_vector_integral_csv_folder_name+".header", 'w') as␣
↪output:
    output.writelines('\n'.join(integral_headers))
```

[178]:
```
def sparse_vector_to_csv_with_nulls_row(additional_column_values, vec,␣
↪num_columns):
    return ','.join([str(value) for value in additional_column_values] +
                    list([ '{:.5}'.format(vec[x]) if x in vec.indices else ''␣
↪for x in range(vec.size) ])[:num_columns]) \
```

```
            .replace('.0,',',')
```

[180]:
```
train_feature_vectors_integral_csv_rdd = train_feature_vectors_exported_df.
↪select(
    'label', 'display_id', 'ad_id', 'document_id', 'document_id_event',␣
↪'feature_vector').withColumn('is_leak', F.lit(-1)) \
    .rdd.map(lambda x: sparse_vector_to_csv_with_nulls_row([x['label'],␣
↪x['display_id'], x['ad_id'], x['document_id'], x['document_id_event'],␣
↪x['is_leak']],
                                                        x['feature_vector'],␣
↪len(integral_headers)))
```

[181]:
```
%time train_feature_vectors_integral_csv_rdd.
↪saveAsTextFile(OUTPUT_BUCKET_FOLDER+train_feature_vector_integral_csv_folder_name)
```

5 Export Validation/Test set feature vectors

[182]:
```
def is_leak(max_timestamp_pv_leak, timestamp_event):
    return max_timestamp_pv_leak >= 0 and max_timestamp_pv_leak >=␣
↪timestamp_event
```

[183]:
```
is_leak_udf = F.udf(lambda max_timestamp_pv_leak, timestamp_event:␣
↪int(is_leak(max_timestamp_pv_leak, timestamp_event)), IntegerType())
```

[184]:
```
if evaluation:
    data_df = validation_set_df
else:
    data_df = test_set_df


test_validation_set_enriched_df = data_df.
↪select('display_id','uuid_event','event_country','event_country_state','platform_event',
                                            'source_id_doc_event',␣
↪'publisher_doc_event','publish_time_doc_event',
                                            'publish_time',
                                          ␣
↪'ad_id','document_id_promo','clicked',
                                           'geo_location_event',␣
↪'advertiser_id', 'publisher_id',
                                           'campaign_id', 'document_id_event',
                                           'traffic_source_pv',
                                        ␣
↪int_list_null_to_empty_list_udf('doc_event_category_id_list').
↪alias('doc_event_category_id_list'),
```

```
                                        ␣
↪float_list_null_to_empty_list_udf('doc_event_confidence_level_cat_list').
↪alias('doc_event_confidence_level_cat_list'),
                                        ␣
↪int_list_null_to_empty_list_udf('doc_event_topic_id_list').
↪alias('doc_event_topic_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('doc_event_confidence_level_top_list').
↪alias('doc_event_confidence_level_top_list'),
                                        ␣
↪str_list_null_to_empty_list_udf('doc_event_entity_id_list').
↪alias('doc_event_entity_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('doc_event_confidence_level_ent_list').
↪alias('doc_event_confidence_level_ent_list'),
                                         int_null_to_minus_one_udf('source_id').
↪alias('source_id'),
                                        ␣
↪int_null_to_minus_one_udf('timestamp_event').alias('timestamp_event'),
                                        ␣
↪int_list_null_to_empty_list_udf('category_id_list').
↪alias('category_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('confidence_level_cat_list').
↪alias('confidence_level_cat_list'),
                                        ␣
↪int_list_null_to_empty_list_udf('topic_id_list').alias('topic_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('confidence_level_top_list').
↪alias('confidence_level_top_list'),
                                        ␣
↪str_list_null_to_empty_list_udf('entity_id_list').alias('entity_id_list'),
                                        ␣
↪float_list_null_to_empty_list_udf('confidence_level_ent_list').
↪alias('confidence_level_ent_list'),
                                        ␣
↪int_null_to_minus_one_udf('max_timestamp_pv').alias('max_timestamp_pv_leak')
                                        ) \
                            .join(user_profiles_df, on=[F.col("user_profiles.
↪uuid") == F.col("uuid_event")], how='left') \
                            .withColumnRenamed('categories', 'user_categories')␣
↪\
                            .withColumnRenamed('topics', 'user_topics') \
                            .withColumnRenamed('entities', 'user_entities') \
                            .withColumnRenamed('doc_ids',␣
↪'user_doc_ids_viewed') \
```

```
                                .withColumnRenamed('views', 'user_views_count')
```

[185]:
```
test_validation_set_feature_vectors_df = test_validation_set_enriched_df \
                                    .withColumn('feature_vector',
                                                #get_ad_feature_vector_udf(
                                                get_ad_feature_vector_integral_udf(
                                                                                   ␣
↪'user_doc_ids_viewed',
                                                                                   ␣
↪'user_views_count',
                                                                                   ␣
↪'user_categories',
                                                                                    'user_topics',
                                                                                    'user_entities',
                                                                                    'event_country',
                                                                                   ␣
↪'event_country_state',
                                                                                    'ad_id',
                                                                                   ␣
↪'document_id_promo',
                                                                                    'source_id',
                                                                                    'publish_time',
                                                                                   ␣
↪'timestamp_event',
                                                                                   ␣
↪'platform_event',
                                                                                   ␣
↪'geo_location_event',
                                                                                   ␣
↪'source_id_doc_event',
                                                                                   ␣
↪'publisher_doc_event',
                                                                                   ␣
↪'publish_time_doc_event',
                                                                                   ␣
↪'traffic_source_pv',
                                                                                    'advertiser_id',
                                                                                    'publisher_id',
                                                                                    'campaign_id',
                                                                                   ␣
↪'document_id_event',
                                                                                   ␣
↪'category_id_list',
                                                                                   ␣
↪'confidence_level_cat_list',
                                                                                    'topic_id_list',
```

```
                                                                    ␣
↪'confidence_level_top_list',
                                                                    ␣
↪'entity_id_list',
                                                                    ␣
↪'confidence_level_ent_list',
                                                                    ␣
↪'doc_event_category_id_list',
                                                                    ␣
↪'doc_event_confidence_level_cat_list',
                                                                    ␣
↪'doc_event_topic_id_list',
                                                                    ␣
↪'doc_event_confidence_level_top_list',
                                                                    ␣
↪'doc_event_entity_id_list',
                                                                    ␣
↪'doc_event_confidence_level_ent_list')) \
                            .select(F.col('uuid').alias('uuid'),
                                    'display_id',
                                    'ad_id',
                                    'document_id_event',
                                    F.col('document_id_promo').
↪alias('document_id'),
                                    F.col('clicked').alias('label'),
                                   ␣
↪is_leak_udf('max_timestamp_pv_leak','timestamp_event').alias('is_leak'),
                                    'feature_vector') #\
                            #.orderBy('display_id','ad_id')
```

[186]:
```
if evaluation:
    test_validation_feature_vector_gcs_folder_name =␣
↪'validation_feature_vectors_integral'
else:
    test_validation_feature_vector_gcs_folder_name =␣
↪'test_feature_vectors_integral'
```

[187]:
```
%time test_validation_set_feature_vectors_df.write.
↪parquet(OUTPUT_BUCKET_FOLDER+test_validation_feature_vector_gcs_folder_name,␣
↪mode='overwrite')
```

5.1 The extraction of the MAP@12 value

The evaluation metric is MAP@12 (Mean Average Precision at 12), which measures the ranking quality.

The expected solution is a CSV file with predictions for test set. The first column was the display_id

and the second column was a ranked list of ad_ids, split by a space character.

[234]:
```
def map_12_a_r(clicked_flag_df,validation_display_ids_df):

    best_params ={'objective':'rank:pairwise',
     'booster': 'gbtree',
     'seed': 'seed',
     'eval_metric': 'map@12',
     'silent': 0,
     'eta': 0.1,
     'max_depth': 11,
     'min_child_weight': 5,
     'gamma': 0.502,
     'colsample_bytree': 0.4,
     'alpha': 1.0,
     'lambda': 30.0,
     'subsample': 1.0
     }
    try:
        num_rounds=18
        clicked_flag = math.sqrt(sum([v**2 for v in clicked_flag_df.values()]))
        validation_display_ids = math.sqrt(sum([v**2 for v in␣
↪validation_display_ids_df.values()]))

        sum_common_aspects = 0.0
        intersections = 0
        for key in clicked_flag_df:
            if key in validation_display_ids:
                sum_common_aspects += clicked_flag_df[key] *␣
↪validation_display_ids_df[key]
                intersections += 1

        watchlist = [((dtrain,'train'), (dvalidation,'validation'))]
        xgb_model = xgb.train(best_params, dtrain, num_boost_round=num_rounds,␣
↪evals=watchlist)
        return sum_common_aspects / (clicked_flag * validation_display_ids),␣
↪intersections

    except:
        gamma=best_params.get('gamma')
        return best_params.get('colsample_bytree')/(best_params.get('lambda')/
↪num_rounds  )+gamma
```

[235]:
```
clicked_flag_df=validation_set_exported_df.select('display_id').distinct().
 ↪createOrReplaceTempView("validation_display_ids")
map_12_a = map_12_a_r(clicked_flag_df,test_validation_set_feature_vectors_df)
map_12_a
```

[235]:
```
0.742
```

5.2 Exporting integral feature vectors to CSV

[210]:
```
test_validation_feature_vectors_exported_df = spark.read.
 ↪parquet(OUTPUT_BUCKET_FOLDER+test_validation_feature_vector_gcs_folder_name)
test_validation_feature_vectors_exported_df.take(3)
```

[211]:
```
if evaluation:
    test_validation_feature_vector_integral_csv_folder_name =␣
 ↪'validation_feature_vectors_integral.csv'
else:
    test_validation_feature_vector_integral_csv_folder_name =␣
 ↪'test_feature_vectors_integral.csv'
```

[212]:
```
integral_headers = ['label', 'display_id', 'ad_id', 'doc_id', 'doc_event_id',␣
 ↪'is_leak'] + feature_vector_labels_integral

with open(test_validation_feature_vector_integral_csv_folder_name+".header",␣
 ↪'w') as output:
    output.writelines('\n'.join(integral_headers))
```

[213]:
```
test_validation_feature_vectors_integral_csv_rdd =␣
 ↪test_validation_feature_vectors_exported_df.select(
    'label', 'display_id', 'ad_id', 'document_id', 'document_id_event',␣
 ↪'is_leak', 'feature_vector') \
    .rdd.map(lambda x: sparse_vector_to_csv_with_nulls_row([x['label'],␣
 ↪x['display_id'], x['ad_id'], x['document_id'], x['document_id_event'],␣
 ↪x['is_leak']],
                                                        x['feature_vector'],␣
 ↪len(integral_headers)))
```