package com.twitter.timelines.prediction.common.aggregates

import com.twitter.conversions.DurationOps.\_

import com.twitter.ml.api.constant.SharedFeatures.AUTHOR\_ID

import com.twitter.ml.api.constant.SharedFeatures.USER\_ID

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.\_

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.metrics.\_

import com.twitter.timelines.data\_processing.ml\_util.transforms.DownsampleTransform

import com.twitter.timelines.data\_processing.ml\_util.transforms.RichRemoveAuthorIdZero

import com.twitter.timelines.data\_processing.ml\_util.transforms.RichRemoveUserIdZero

import com.twitter.timelines.prediction.features.common.TimelinesSharedFeatures

import com.twitter.timelines.prediction.features.engagement\_features.EngagementDataRecordFeatures

import com.twitter.timelines.prediction.features.engagement\_features.EngagementDataRecordFeatures.RichUnifyPublicEngagersTransform

import com.twitter.timelines.prediction.features.list\_features.ListFeatures

import com.twitter.timelines.prediction.features.recap.RecapFeatures

import com.twitter.timelines.prediction.features.request\_context.RequestContextFeatures

import com.twitter.timelines.prediction.features.semantic\_core\_features.SemanticCoreFeatures

import com.twitter.timelines.prediction.transform.filter.FilterInNetworkTransform

import com.twitter.timelines.prediction.transform.filter.FilterImageTweetTransform

import com.twitter.timelines.prediction.transform.filter.FilterVideoTweetTransform

import com.twitter.timelines.prediction.transform.filter.FilterOutImageVideoTweetTransform

import com.twitter.util.Duration

trait TimelinesAggregationConfigDetails extends Serializable {

import TimelinesAggregationSources.\_

def outputHdfsPath: String

/\*\*

\* Converts the given logical store to a physical store. The reason we do not specify the

\* physical store directly with the [[AggregateGroup]] is because of a cyclic dependency when

\* create physical stores that are DalDataset with PersonalDataType annotations derived from

\* the [[AggregateGroup]].

\*

\*/

def mkPhysicalStore(store: AggregateStore): AggregateStore

def defaultMaxKvSourceFailures: Int = 100

val timelinesOfflineAggregateSink = new OfflineStoreCommonConfig {

override def apply(startDate: String) = OfflineAggregateStoreCommonConfig(

outputHdfsPathPrefix = outputHdfsPath,

dummyAppId = "timelines\_aggregates\_v2\_ro",

dummyDatasetPrefix = "timelines\_aggregates\_v2\_ro",

startDate = startDate

)

}

val UserAggregateStore = "user\_aggregates"

val UserAuthorAggregateStore = "user\_author\_aggregates"

val UserOriginalAuthorAggregateStore = "user\_original\_author\_aggregates"

val OriginalAuthorAggregateStore = "original\_author\_aggregates"

val UserEngagerAggregateStore = "user\_engager\_aggregates"

val UserMentionAggregateStore = "user\_mention\_aggregates"

val TwitterWideUserAggregateStore = "twitter\_wide\_user\_aggregates"

val TwitterWideUserAuthorAggregateStore = "twitter\_wide\_user\_author\_aggregates"

val UserRequestHourAggregateStore = "user\_request\_hour\_aggregates"

val UserRequestDowAggregateStore = "user\_request\_dow\_aggregates"

val UserListAggregateStore = "user\_list\_aggregates"

val AuthorTopicAggregateStore = "author\_topic\_aggregates"

val UserTopicAggregateStore = "user\_topic\_aggregates"

val UserInferredTopicAggregateStore = "user\_inferred\_topic\_aggregates"

val UserMediaUnderstandingAnnotationAggregateStore =

"user\_media\_understanding\_annotation\_aggregates"

val AuthorCountryCodeAggregateStore = "author\_country\_code\_aggregates"

val OriginalAuthorCountryCodeAggregateStore = "original\_author\_country\_code\_aggregates"

/\*\*

\* Step 3: Configure all aggregates to compute.

\* Note that different subsets of aggregates in this list

\* can be launched by different summingbird job instances.

\* Any given job can be responsible for a set of AggregateGroup

\* configs whose outputStores share the same exact startDate.

\* AggregateGroups that do not share the same inputSource,

\* outputStore or startDate MUST be launched using different

\* summingbird jobs and passed in a different --start-time argument

\* See science/scalding/mesos/timelines/prod.yaml for an example

\* of how to configure your own job.

\*/

val negativeDownsampleTransform =

DownsampleTransform(

negativeSamplingRate = 0.03,

keepLabels = RecapUserFeatureAggregation.LabelsV2)

val negativeRecTweetDownsampleTransform = DownsampleTransform(

negativeSamplingRate = 0.03,

keepLabels = RectweetUserFeatureAggregation.RectweetLabelsForAggregation

)

val userAggregatesV2: AggregateGroup =

AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_aggregate\_v2",

preTransforms = Seq(RichRemoveUserIdZero), /\* Eliminates reducer skew \*/

keys = Set(USER\_ID),

features = RecapUserFeatureAggregation.UserFeaturesV2,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric, SumMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userAuthorAggregatesV2: Set[AggregateGroup] = {

/\*\*

\* NOTE: We need to remove records from out-of-network authors from the recap input

\* records (which now include out-of-network records as well after merging recap and

\* rectweet models) that are used to compute user-author aggregates. This is necessary

\* to limit the growth rate of user-author aggregates.

\*/

val allFeatureAggregates = Set(

AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_author\_aggregate\_v2",

preTransforms = Seq(FilterInNetworkTransform, RichRemoveUserIdZero),

keys = Set(USER\_ID, AUTHOR\_ID),

features = RecapUserFeatureAggregation.UserAuthorFeaturesV2,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(SumMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAuthorAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

)

val countAggregates: Set[AggregateGroup] = Set(

AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_author\_aggregate\_v2",

preTransforms = Seq(FilterInNetworkTransform, RichRemoveUserIdZero),

keys = Set(USER\_ID, AUTHOR\_ID),

features = RecapUserFeatureAggregation.UserAuthorFeaturesV2Count,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAuthorAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

)

allFeatureAggregates ++ countAggregates

}

val userAggregatesV5Continuous: AggregateGroup =

AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_aggregate\_v5.continuous",

preTransforms = Seq(RichRemoveUserIdZero),

keys = Set(USER\_ID),

features = RecapUserFeatureAggregation.UserFeaturesV5Continuous,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric, SumMetric, SumSqMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userAuthorAggregatesV5: AggregateGroup =

AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_author\_aggregate\_v5",

preTransforms = Seq(FilterInNetworkTransform, RichRemoveUserIdZero),

keys = Set(USER\_ID, AUTHOR\_ID),

features = RecapUserFeatureAggregation.UserAuthorFeaturesV5,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAuthorAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val tweetSourceUserAuthorAggregatesV1: AggregateGroup =

AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_author\_aggregate\_tweetsource\_v1",

preTransforms = Seq(FilterInNetworkTransform, RichRemoveUserIdZero),

keys = Set(USER\_ID, AUTHOR\_ID),

features = RecapUserFeatureAggregation.UserAuthorTweetSourceFeaturesV1,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric, SumMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAuthorAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userEngagerAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_engager\_aggregate",

keys = Set(USER\_ID, EngagementDataRecordFeatures.PublicEngagementUserIds),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserEngagerAggregateStore,

startDate = "2016-09-02 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

preTransforms = Seq(

RichRemoveUserIdZero,

RichUnifyPublicEngagersTransform

)

)

val userMentionAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

preTransforms = Seq(RichRemoveUserIdZero), /\* Eliminates reducer skew \*/

aggregatePrefix = "user\_mention\_aggregate",

keys = Set(USER\_ID, RecapFeatures.MENTIONED\_SCREEN\_NAMES),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserMentionAggregateStore,

startDate = "2017-03-01 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

includeAnyLabel = false

)

val twitterWideUserAggregates = AggregateGroup(

inputSource = timelinesDailyTwitterWideSource,

preTransforms = Seq(RichRemoveUserIdZero), /\* Eliminates reducer skew \*/

aggregatePrefix = "twitter\_wide\_user\_aggregate",

keys = Set(USER\_ID),

features = RecapUserFeatureAggregation.TwitterWideFeatures,

labels = RecapUserFeatureAggregation.TwitterWideLabels,

metrics = Set(CountMetric, SumMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = TwitterWideUserAggregateStore,

startDate = "2016-12-28 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val twitterWideUserAuthorAggregates = AggregateGroup(

inputSource = timelinesDailyTwitterWideSource,

preTransforms = Seq(RichRemoveUserIdZero), /\* Eliminates reducer skew \*/

aggregatePrefix = "twitter\_wide\_user\_author\_aggregate",

keys = Set(USER\_ID, AUTHOR\_ID),

features = RecapUserFeatureAggregation.TwitterWideFeatures,

labels = RecapUserFeatureAggregation.TwitterWideLabels,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = TwitterWideUserAuthorAggregateStore,

startDate = "2016-12-28 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

includeAnyLabel = false

)

/\*\*

\* User-HourOfDay and User-DayOfWeek aggregations, both for recap and rectweet

\*/

val userRequestHourAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_request\_context\_aggregate.hour",

preTransforms = Seq(RichRemoveUserIdZero, negativeDownsampleTransform),

keys = Set(USER\_ID, RequestContextFeatures.TIMESTAMP\_GMT\_HOUR),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserRequestHourAggregateStore,

startDate = "2017-08-01 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userRequestDowAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_request\_context\_aggregate.dow",

preTransforms = Seq(RichRemoveUserIdZero, negativeDownsampleTransform),

keys = Set(USER\_ID, RequestContextFeatures.TIMESTAMP\_GMT\_DOW),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserRequestDowAggregateStore,

startDate = "2017-08-01 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val authorTopicAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "author\_topic\_aggregate",

preTransforms = Seq(RichRemoveUserIdZero),

keys = Set(AUTHOR\_ID, TimelinesSharedFeatures.TOPIC\_ID),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = AuthorTopicAggregateStore,

startDate = "2020-05-19 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userTopicAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_topic\_aggregate",

preTransforms = Seq(RichRemoveUserIdZero),

keys = Set(USER\_ID, TimelinesSharedFeatures.TOPIC\_ID),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserTopicAggregateStore,

startDate = "2020-05-23 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userTopicAggregatesV2 = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_topic\_aggregate\_v2",

preTransforms = Seq(RichRemoveUserIdZero),

keys = Set(USER\_ID, TimelinesSharedFeatures.TOPIC\_ID),

features = RecapUserFeatureAggregation.UserTopicFeaturesV2Count,

labels = RecapUserFeatureAggregation.LabelsV2,

includeAnyFeature = false,

includeAnyLabel = false,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserTopicAggregateStore,

startDate = "2020-05-23 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userInferredTopicAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_inferred\_topic\_aggregate",

preTransforms = Seq(RichRemoveUserIdZero),

keys = Set(USER\_ID, TimelinesSharedFeatures.INFERRED\_TOPIC\_IDS),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserInferredTopicAggregateStore,

startDate = "2020-09-09 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userInferredTopicAggregatesV2 = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_inferred\_topic\_aggregate\_v2",

preTransforms = Seq(RichRemoveUserIdZero),

keys = Set(USER\_ID, TimelinesSharedFeatures.INFERRED\_TOPIC\_IDS),

features = RecapUserFeatureAggregation.UserTopicFeaturesV2Count,

labels = RecapUserFeatureAggregation.LabelsV2,

includeAnyFeature = false,

includeAnyLabel = false,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserInferredTopicAggregateStore,

startDate = "2020-09-09 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userReciprocalEngagementAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_aggregate\_v6",

preTransforms = Seq(RichRemoveUserIdZero),

keys = Set(USER\_ID),

features = Set.empty,

labels = RecapUserFeatureAggregation.ReciprocalLabels,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

includeAnyLabel = false

)

val userOriginalAuthorReciprocalEngagementAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_original\_author\_aggregate\_v1",

preTransforms = Seq(RichRemoveUserIdZero, RichRemoveAuthorIdZero),

keys = Set(USER\_ID, TimelinesSharedFeatures.ORIGINAL\_AUTHOR\_ID),

features = Set.empty,

labels = RecapUserFeatureAggregation.ReciprocalLabels,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserOriginalAuthorAggregateStore,

startDate = "2018-12-26 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

includeAnyLabel = false

)

val originalAuthorReciprocalEngagementAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "original\_author\_aggregate\_v1",

preTransforms = Seq(RichRemoveUserIdZero, RichRemoveAuthorIdZero),

keys = Set(TimelinesSharedFeatures.ORIGINAL\_AUTHOR\_ID),

features = Set.empty,

labels = RecapUserFeatureAggregation.ReciprocalLabels,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = OriginalAuthorAggregateStore,

startDate = "2023-02-25 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

includeAnyLabel = false

)

val originalAuthorNegativeEngagementAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "original\_author\_aggregate\_v2",

preTransforms = Seq(RichRemoveUserIdZero, RichRemoveAuthorIdZero),

keys = Set(TimelinesSharedFeatures.ORIGINAL\_AUTHOR\_ID),

features = Set.empty,

labels = RecapUserFeatureAggregation.NegativeEngagementLabels,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = OriginalAuthorAggregateStore,

startDate = "2023-02-25 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

includeAnyLabel = false

)

val userListAggregates: AggregateGroup =

AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_list\_aggregate",

keys = Set(USER\_ID, ListFeatures.LIST\_ID),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserListAggregateStore,

startDate = "2020-05-28 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

preTransforms = Seq(RichRemoveUserIdZero)

)

val userMediaUnderstandingAnnotationAggregates: AggregateGroup = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_media\_annotation\_aggregate",

preTransforms = Seq(RichRemoveUserIdZero),

keys =

Set(USER\_ID, SemanticCoreFeatures.mediaUnderstandingHighRecallNonSensitiveEntityIdsFeature),

features = Set.empty,

labels = RecapUserFeatureAggregation.LabelsV2,

metrics = Set(CountMetric),

halfLives = Set(50.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserMediaUnderstandingAnnotationAggregateStore,

startDate = "2021-03-20 00:00",

commonConfig = timelinesOfflineAggregateSink

))

)

val userAuthorGoodClickAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_author\_good\_click\_aggregate",

preTransforms = Seq(FilterInNetworkTransform, RichRemoveUserIdZero),

keys = Set(USER\_ID, AUTHOR\_ID),

features = RecapUserFeatureAggregation.UserAuthorFeaturesV2,

labels = RecapUserFeatureAggregation.GoodClickLabels,

metrics = Set(SumMetric),

halfLives = Set(14.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserAuthorAggregateStore,

startDate = "2016-07-15 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

))

)

val userEngagerGoodClickAggregates = AggregateGroup(

inputSource = timelinesDailyRecapMinimalSource,

aggregatePrefix = "user\_engager\_good\_click\_aggregate",

keys = Set(USER\_ID, EngagementDataRecordFeatures.PublicEngagementUserIds),

features = Set.empty,

labels = RecapUserFeatureAggregation.GoodClickLabels,

metrics = Set(CountMetric),

halfLives = Set(14.days),

outputStore = mkPhysicalStore(

OfflineAggregateDataRecordStore(

name = UserEngagerAggregateStore,

startDate = "2016-09-02 00:00",

commonConfig = timelinesOfflineAggregateSink,

maxKvSourceFailures = defaultMaxKvSourceFailures

)),

preTransforms = Seq(

RichRemoveUserIdZero,

RichUnifyPublicEngagersTransform

)

)

}