package com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework

import com.twitter.ml.api.constant.SharedFeatures

import com.twitter.ml.api.Feature

import com.twitter.ml.api.FeatureType

/\*\*

\* Convenience class to describe the stores that make up a particular type of aggregate.

\*

\* For example, as of 2018/07, user aggregates are generate by merging the individual

\* "user\_aggregates", "rectweet\_user\_aggregates", and, "twitter\_wide\_user\_aggregates".

\*

\* @param storeNames Name of the stores.

\* @param aggregateType Type of aggregate, usually differentiated by the aggregation key.

\* @param shouldHash Used at TimelineRankingAggregatesUtil.extractSecondary when extracting the

\* secondary key value.

\*/

case class StoreConfig[T](

storeNames: Set[String],

aggregateType: AggregateType.Value,

shouldHash: Boolean = false

)(

implicit storeMerger: StoreMerger) {

require(storeMerger.isValidToMerge(storeNames))

private val representativeStore = storeNames.head

val aggregationKeyIds: Set[Long] = storeMerger.getAggregateKeys(representativeStore)

val aggregationKeyFeatures: Set[Feature[\_]] =

storeMerger.getAggregateKeyFeatures(representativeStore)

val secondaryKeyFeatureOpt: Option[Feature[\_]] = storeMerger.getSecondaryKey(representativeStore)

}

trait StoreMerger {

def aggregationConfig: AggregationConfig

def getAggregateKeyFeatures(storeName: String): Set[Feature[\_]] =

aggregationConfig.aggregatesToCompute

.filter(\_.outputStore.name == storeName)

.flatMap(\_.keysToAggregate)

def getAggregateKeys(storeName: String): Set[Long] =

TypedAggregateGroup.getKeyFeatureIds(getAggregateKeyFeatures(storeName))

def getSecondaryKey(storeName: String): Option[Feature[\_]] = {

val keys = getAggregateKeyFeatures(storeName)

require(keys.size <= 2, "Only singleton or binary aggregation keys are supported.")

require(keys.contains(SharedFeatures.USER\_ID), "USER\_ID must be one of the aggregation keys.")

keys

.filterNot(\_ == SharedFeatures.USER\_ID)

.headOption

.map { possiblySparseKey =>

if (possiblySparseKey.getFeatureType != FeatureType.SPARSE\_BINARY) {

possiblySparseKey

} else {

TypedAggregateGroup.sparseFeature(possiblySparseKey)

}

}

}

/\*\*

\* Stores may only be merged if they have the same aggregation key.

\*/

def isValidToMerge(storeNames: Set[String]): Boolean = {

val expectedKeyOpt = storeNames.headOption.map(getAggregateKeys)

storeNames.forall(v => getAggregateKeys(v) == expectedKeyOpt.get)

}

}