package com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.author\_features

import com.twitter.home\_mixer.util.DataRecordUtil

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.Feature

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.util.CompactDataRecordConverter

import com.twitter.ml.api.util.FDsl.\_

import com.twitter.timelines.author\_features.v1.{thriftjava => af}

import com.twitter.timelines.prediction.common.adapters.TimelinesAdapterBase

import com.twitter.timelines.prediction.common.aggregates.TimelinesAggregationConfig

import com.twitter.timelines.prediction.features.user\_health.UserHealthFeatures

import scala.collection.JavaConverters.\_

object AuthorFeaturesAdapter extends TimelinesAdapterBase[af.AuthorFeatures] {

private val Prefix = "original\_author.timelines.original\_author\_aggregates."

private val typedAggregateGroups =

TimelinesAggregationConfig.originalAuthorAggregatesV1.buildTypedAggregateGroups()

private val aggregateFeaturesRenameMap: Map[Feature[\_], Feature[\_]] =

typedAggregateGroups.map(\_.outputFeaturesToRenamedOutputFeatures(Prefix)).reduce(\_ ++ \_)

private val prefixedOriginalAuthorAggregateFeatures =

typedAggregateGroups.flatMap(\_.allOutputFeatures).map { feature =>

aggregateFeaturesRenameMap.getOrElse(feature, feature)

}

private val authorFeatures = prefixedOriginalAuthorAggregateFeatures ++ Seq(

UserHealthFeatures.AuthorState,

UserHealthFeatures.NumAuthorFollowers,

UserHealthFeatures.NumAuthorConnectDays,

UserHealthFeatures.NumAuthorConnect

)

private val aggregateFeatureContext: FeatureContext =

new FeatureContext(typedAggregateGroups.flatMap(\_.allOutputFeatures).asJava)

private lazy val prefixedAggregateFeatureContext: FeatureContext =

new FeatureContext(prefixedOriginalAuthorAggregateFeatures.asJava)

override val getFeatureContext: FeatureContext = new FeatureContext(authorFeatures: \_\*)

override val commonFeatures: Set[Feature[\_]] = Set.empty

private val compactDataRecordConverter = new CompactDataRecordConverter()

override def adaptToDataRecords(

authorFeatures: af.AuthorFeatures

): java.util.List[DataRecord] = {

val dataRecord =

if (authorFeatures.aggregates != null) {

val originalAuthorAggregatesDataRecord =

compactDataRecordConverter.compactDataRecordToDataRecord(authorFeatures.aggregates)

DataRecordUtil.applyRename(

originalAuthorAggregatesDataRecord,

aggregateFeatureContext,

prefixedAggregateFeatureContext,

aggregateFeaturesRenameMap)

} else new DataRecord

if (authorFeatures.user\_health != null) {

val userHealth = authorFeatures.user\_health

if (userHealth.user\_state != null) {

dataRecord.setFeatureValue(

UserHealthFeatures.AuthorState,

userHealth.user\_state.getValue.toLong

)

}

dataRecord.setFeatureValue(

UserHealthFeatures.NumAuthorFollowers,

userHealth.num\_followers.toDouble

)

dataRecord.setFeatureValue(

UserHealthFeatures.NumAuthorConnectDays,

userHealth.num\_connect\_days.toDouble

)

dataRecord.setFeatureValue(

UserHealthFeatures.NumAuthorConnect,

userHealth.num\_connect.toDouble

)

}

List(dataRecord).asJava

}

}