package com.twitter.follow\_recommendations.common.feature\_hydration.sources

import com.google.inject.Inject

import com.google.inject.Provides

import com.google.inject.Singleton

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.feature\_hydration.adapters.CandidateAlgorithmAdapter

import com.twitter.follow\_recommendations.common.feature\_hydration.common.FeatureSource

import com.twitter.follow\_recommendations.common.feature\_hydration.common.FeatureSourceId

import com.twitter.follow\_recommendations.common.feature\_hydration.common.HasPreFetchedFeature

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.models.HasDisplayLocation

import com.twitter.follow\_recommendations.common.models.HasSimilarToContext

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.FeatureContext

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

/\*\*

\* This source only takes features from the candidate's source,

\* which is all the information we have about the candidate pre-feature-hydration

\*/

@Provides

@Singleton

class CandidateAlgorithmSource @Inject() (stats: StatsReceiver) extends FeatureSource {

override val id: FeatureSourceId = FeatureSourceId.CandidateAlgorithmSourceId

override val featureContext: FeatureContext = CandidateAlgorithmAdapter.getFeatureContext

override def hydrateFeatures(

t: HasClientContext

with HasPreFetchedFeature

with HasParams

with HasSimilarToContext

with HasDisplayLocation, // we don't use the target here

candidates: Seq[CandidateUser]

): Stitch[Map[CandidateUser, DataRecord]] = {

val featureHydrationStats = stats.scope("candidate\_alg\_source")

val hasSourceDetailsStat = featureHydrationStats.counter("has\_source\_details")

val noSourceDetailsStat = featureHydrationStats.counter("no\_source\_details")

val noSourceRankStat = featureHydrationStats.counter("no\_source\_rank")

val hasSourceRankStat = featureHydrationStats.counter("has\_source\_rank")

val noSourceScoreStat = featureHydrationStats.counter("no\_source\_score")

val hasSourceScoreStat = featureHydrationStats.counter("has\_source\_score")

val candidatesToAlgoMap = for {

candidate <- candidates

} yield {

if (candidate.userCandidateSourceDetails.nonEmpty) {

hasSourceDetailsStat.incr()

candidate.userCandidateSourceDetails.foreach { details =>

if (details.candidateSourceRanks.isEmpty) {

noSourceRankStat.incr()

} else {

hasSourceRankStat.incr()

}

if (details.candidateSourceScores.isEmpty) {

noSourceScoreStat.incr()

} else {

hasSourceScoreStat.incr()

}

}

} else {

noSourceDetailsStat.incr()

}

candidate -> CandidateAlgorithmAdapter.adaptToDataRecord(candidate.userCandidateSourceDetails)

}

Stitch.value(candidatesToAlgoMap.toMap)

}

}