package com.twitter.follow\_recommendations.common.rankers.ml\_ranker.ranking

import com.google.inject.Inject

import com.google.inject.Singleton

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.GatedTransform

import com.twitter.follow\_recommendations.common.base.StatsUtil.profileStitchMapResults

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

import com.twitter.follow\_recommendations.common.feature\_hydration.sources.UserScoringFeatureSource

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

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

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

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

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

import com.twitter.ml.api.DataRecord

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

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import com.twitter.util.logging.Logging

/\*\*

\* Hydrate features given target and candidates lists.

\* This is a required step before MlRanker.

\* If a feature is not hydrated before MlRanker is triggered, a runtime exception will be thrown

\*/

@Singleton

class HydrateFeaturesTransform[

Target <: HasClientContext with HasParams with HasDebugOptions with HasPreFetchedFeature with HasSimilarToContext with HasDisplayLocation] @Inject() (

userScoringFeatureSource: UserScoringFeatureSource,

stats: StatsReceiver)

extends GatedTransform[Target, CandidateUser]

with Logging {

private val hydrateFeaturesStats = stats.scope("hydrate\_features")

def transform(target: Target, candidates: Seq[CandidateUser]): Stitch[Seq[CandidateUser]] = {

// get features

val featureMapStitch: Stitch[Map[CandidateUser, DataRecord]] =

profileStitchMapResults(

userScoringFeatureSource.hydrateFeatures(target, candidates),

hydrateFeaturesStats)

featureMapStitch.map { featureMap =>

candidates

.map { candidate =>

val dataRecord = featureMap(candidate)

// add debugRecord only when the request parameter is set

val debugDataRecord = if (target.debugOptions.exists(\_.fetchDebugInfo)) {

Some(candidate.toDebugDataRecord(dataRecord, userScoringFeatureSource.featureContext))

} else None

candidate.copy(

dataRecord = Some(RichDataRecord(Some(dataRecord), debugDataRecord))

)

}

}

}

}