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

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

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

import com.twitter.ml.api.Feature.Continuous

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

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.IRecordOneToOneAdapter

import com.twitter.util.Time

/\*\*

\* This adapter mimics UserRecentWTFImpressionsAndFollowsAdapter (for user) and

\* RecentWTFImpressionsFeatureAdapter (for candidate) for extracting recent impression

\* and follow features. This adapter extracts user, candidate, and pair-wise features.

\*/

object PreFetchedFeatureAdapter

extends IRecordOneToOneAdapter[

(HasPreFetchedFeature, CandidateUser)

] {

// impression features

val USER\_NUM\_RECENT\_IMPRESSIONS: Continuous = new Continuous(

"user.prefetch.num\_recent\_impressions"

)

val USER\_LAST\_IMPRESSION\_DURATION: Continuous = new Continuous(

"user.prefetch.last\_impression\_duration"

)

val CANDIDATE\_NUM\_RECENT\_IMPRESSIONS: Continuous = new Continuous(

"user-candidate.prefetch.num\_recent\_impressions"

)

val CANDIDATE\_LAST\_IMPRESSION\_DURATION: Continuous = new Continuous(

"user-candidate.prefetch.last\_impression\_duration"

)

// follow features

val USER\_NUM\_RECENT\_FOLLOWERS: Continuous = new Continuous(

"user.prefetch.num\_recent\_followers"

)

val USER\_NUM\_RECENT\_FOLLOWED\_BY: Continuous = new Continuous(

"user.prefetch.num\_recent\_followed\_by"

)

val USER\_NUM\_RECENT\_MUTUAL\_FOLLOWS: Continuous = new Continuous(

"user.prefetch.num\_recent\_mutual\_follows"

)

// impression + follow features

val USER\_NUM\_RECENT\_FOLLOWED\_IMPRESSIONS: Continuous = new Continuous(

"user.prefetch.num\_recent\_followed\_impression"

)

val USER\_LAST\_FOLLOWED\_IMPRESSION\_DURATION: Continuous = new Continuous(

"user.prefetch.last\_followed\_impression\_duration"

)

override def adaptToDataRecord(

record: (HasPreFetchedFeature, CandidateUser)

): DataRecord = {

val (target, candidate) = record

val dr = new DataRecord()

val t = Time.now

// set impression features for user, optionally for candidate

dr.setFeatureValue(USER\_NUM\_RECENT\_IMPRESSIONS, target.numWtfImpressions.toDouble)

dr.setFeatureValue(

USER\_LAST\_IMPRESSION\_DURATION,

(t - target.latestImpressionTime).inMillis.toDouble)

target.getCandidateImpressionCounts(candidate.id).foreach { counts =>

dr.setFeatureValue(CANDIDATE\_NUM\_RECENT\_IMPRESSIONS, counts.toDouble)

}

target.getCandidateLatestTime(candidate.id).foreach { latestTime: Time =>

dr.setFeatureValue(CANDIDATE\_LAST\_IMPRESSION\_DURATION, (t - latestTime).inMillis.toDouble)

}

// set recent follow features for user

dr.setFeatureValue(USER\_NUM\_RECENT\_FOLLOWERS, target.numRecentFollowedUserIds.toDouble)

dr.setFeatureValue(USER\_NUM\_RECENT\_FOLLOWED\_BY, target.numRecentFollowedByUserIds.toDouble)

dr.setFeatureValue(USER\_NUM\_RECENT\_MUTUAL\_FOLLOWS, target.numRecentMutualFollows.toDouble)

dr.setFeatureValue(USER\_NUM\_RECENT\_FOLLOWED\_IMPRESSIONS, target.numFollowedImpressions.toDouble)

dr.setFeatureValue(

USER\_LAST\_FOLLOWED\_IMPRESSION\_DURATION,

target.lastFollowedImpressionDurationMs.getOrElse(Long.MaxValue).toDouble)

dr

}

override def getFeatureContext: FeatureContext = new FeatureContext(

USER\_NUM\_RECENT\_IMPRESSIONS,

USER\_LAST\_IMPRESSION\_DURATION,

CANDIDATE\_NUM\_RECENT\_IMPRESSIONS,

CANDIDATE\_LAST\_IMPRESSION\_DURATION,

USER\_NUM\_RECENT\_FOLLOWERS,

USER\_NUM\_RECENT\_FOLLOWED\_BY,

USER\_NUM\_RECENT\_MUTUAL\_FOLLOWS,

USER\_NUM\_RECENT\_FOLLOWED\_IMPRESSIONS,

USER\_LAST\_FOLLOWED\_IMPRESSION\_DURATION,

)

}