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

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

import com.twitter.ml.api.Feature.Binary

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

import com.twitter.ml.api.Feature.Discrete

import com.twitter.ml.api.Feature.Text

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.onboarding.relevance.util.metadata.LanguageUtil

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

import com.twitter.snowflake.id.SnowflakeId

object ClientContextAdapter extends IRecordOneToOneAdapter[(ClientContext, DisplayLocation)] {

// we name features with `user.account` for relatively static user-related features

val USER\_COUNTRY: Text = new Text("user.account.country")

val USER\_LANGUAGE: Text = new Text("user.account.language")

// we name features with `user.context` for more dynamic user-related features

val USER\_LANGUAGE\_PREFIX: Text = new Text("user.context.language\_prefix")

val USER\_CLIENT: Discrete = new Discrete("user.context.client")

val USER\_AGE: Continuous = new Continuous("user.context.age")

val USER\_IS\_RECENT: Binary = new Binary("user.is.recent")

// we name features with `meta` for meta info about the WTF recommendation request

val META\_DISPLAY\_LOCATION: Text = new Text("meta.display\_location")

val META\_POSITION: Discrete = new Discrete("meta.position")

// This indicates whether a data point is from a random serving policy

val META\_IS\_RANDOM: Binary = new Binary("prediction.engine.is\_random")

val RECENT\_WIN\_IN\_DAYS: Int = 30

val GOAL\_META\_POSITION: Long = 1L

val GOAL\_META\_IS\_RANDOM: Boolean = true

override val getFeatureContext: FeatureContext = new FeatureContext(

USER\_COUNTRY,

USER\_LANGUAGE,

USER\_AGE,

USER\_LANGUAGE\_PREFIX,

USER\_CLIENT,

USER\_IS\_RECENT,

META\_DISPLAY\_LOCATION,

META\_POSITION,

META\_IS\_RANDOM

)

/\*\*

\* we only want to set the relevant fields iff they exist to eliminate redundant information

\* we do some simple normalization on the language code

\* we set META\_POSITION to 1 always

\* we set META\_IS\_RANDOM to true always to simulate a random serving distribution

\* @param record ClientContext and DisplayLocation from the request

\*/

override def adaptToDataRecord(target: (ClientContext, DisplayLocation)): DataRecord = {

val dr = new DataRecord()

val cc = target.\_1

val dl = target.\_2

cc.countryCode.foreach(countryCode => dr.setFeatureValue(USER\_COUNTRY, countryCode))

cc.languageCode.foreach(rawLanguageCode => {

val userLanguage = LanguageUtil.simplifyLanguage(rawLanguageCode)

val userLanguagePrefix = userLanguage.take(2)

dr.setFeatureValue(USER\_LANGUAGE, userLanguage)

dr.setFeatureValue(USER\_LANGUAGE\_PREFIX, userLanguagePrefix)

})

cc.appId.foreach(appId => dr.setFeatureValue(USER\_CLIENT, appId))

cc.userId.foreach(id =>

SnowflakeId.timeFromIdOpt(id).map { signupTime =>

val userAge = signupTime.untilNow.inMillis.toDouble

dr.setFeatureValue(USER\_AGE, userAge)

dr.setFeatureValue(USER\_IS\_RECENT, signupTime.untilNow.inDays <= RECENT\_WIN\_IN\_DAYS)

signupTime.untilNow.inDays

})

dr.setFeatureValue(META\_DISPLAY\_LOCATION, dl.toFsName)

dr.setFeatureValue(META\_POSITION, GOAL\_META\_POSITION)

dr.setFeatureValue(META\_IS\_RANDOM, GOAL\_META\_IS\_RANDOM)

dr

}

}