package com.twitter.timelines.prediction.common.aggregates.real\_time

import com.twitter.dal.personal\_data.thriftjava.PersonalDataType.InferredGender

import com.twitter.dal.personal\_data.thriftjava.PersonalDataType.UserState

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

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

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.Feature

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.RichDataRecord

import com.twitter.ml.featurestore.catalog.entities.core.User

import com.twitter.ml.featurestore.catalog.features.core.UserAccount

import com.twitter.ml.featurestore.catalog.features.geo.UserLocation

import com.twitter.ml.featurestore.catalog.features.magicrecs.UserActivity

import com.twitter.ml.featurestore.lib.EntityId

import com.twitter.ml.featurestore.lib.data.PredictionRecord

import com.twitter.ml.featurestore.lib.feature.BoundFeature

import com.twitter.ml.featurestore.lib.feature.BoundFeatureSet

import com.twitter.ml.featurestore.lib.UserId

import com.twitter.ml.featurestore.lib.{Discrete => FSDiscrete}

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

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

import java.lang.{Boolean => JBoolean}

import java.lang.{String => JString}

import java.util

import scala.collection.JavaConverters.\_

object UserFeaturesAdapter extends TimelinesAdapterBase[PredictionRecord] {

val UserStateBoundFeature: BoundFeature[UserId, FSDiscrete] = UserActivity.UserState.bind(User)

/\*\*

\* Boolean features about viewer's user state.

\* enum UserState {

\* NEW = 0,

\* NEAR\_ZERO = 1,

\* VERY\_LIGHT = 2,

\* LIGHT = 3,

\* MEDIUM\_TWEETER = 4,

\* MEDIUM\_NON\_TWEETER = 5,

\* HEAVY\_NON\_TWEETER = 6,

\* HEAVY\_TWEETER = 7

\* }(persisted='true')

\*/

val IS\_USER\_NEW = new Binary("timelines.user\_state.is\_user\_new", Set(UserState).asJava)

val IS\_USER\_LIGHT = new Binary("timelines.user\_state.is\_user\_light", Set(UserState).asJava)

val IS\_USER\_MEDIUM\_TWEETER =

new Binary("timelines.user\_state.is\_user\_medium\_tweeter", Set(UserState).asJava)

val IS\_USER\_MEDIUM\_NON\_TWEETER =

new Binary("timelines.user\_state.is\_user\_medium\_non\_tweeter", Set(UserState).asJava)

val IS\_USER\_HEAVY\_NON\_TWEETER =

new Binary("timelines.user\_state.is\_user\_heavy\_non\_tweeter", Set(UserState).asJava)

val IS\_USER\_HEAVY\_TWEETER =

new Binary("timelines.user\_state.is\_user\_heavy\_tweeter", Set(UserState).asJava)

val userStateToFeatureMap: Map[Long, Binary] = Map(

0L -> IS\_USER\_NEW,

1L -> IS\_USER\_LIGHT,

2L -> IS\_USER\_LIGHT,

3L -> IS\_USER\_LIGHT,

4L -> IS\_USER\_MEDIUM\_TWEETER,

5L -> IS\_USER\_MEDIUM\_NON\_TWEETER,

6L -> IS\_USER\_HEAVY\_NON\_TWEETER,

7L -> IS\_USER\_HEAVY\_TWEETER

)

val UserStateBooleanFeatures: Set[Feature[\_]] = userStateToFeatureMap.values.toSet

val USER\_COUNTRY\_ID = new Text("geo.user\_location.country\_code")

val UserCountryCodeFeature: BoundFeature[UserId, String] =

UserLocation.CountryCodeAlpha2.bind(User)

val UserLocationFeatures: Set[Feature[\_]] = Set(USER\_COUNTRY\_ID)

private val UserVerifiedFeaturesSet = Set(

UserAccount.IsUserVerified.bind(User),

UserAccount.IsUserBlueVerified.bind(User),

UserAccount.IsUserGoldVerified.bind(User),

UserAccount.IsUserGrayVerified.bind(User)

)

val UserFeaturesSet: BoundFeatureSet =

BoundFeatureSet(UserStateBoundFeature, UserCountryCodeFeature) ++

BoundFeatureSet(UserVerifiedFeaturesSet.asInstanceOf[Set[BoundFeature[\_ <: EntityId, \_]]])

private val allFeatures: Seq[Feature[\_]] =

UserStateBooleanFeatures.toSeq ++ GenderBooleanFeatures.toSeq ++

UserLocationFeatures.toSeq ++ Seq(UserHealthFeatures.IsUserVerifiedUnion)

override def getFeatureContext: FeatureContext = new FeatureContext(allFeatures: \_\*)

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

override def adaptToDataRecords(record: PredictionRecord): util.List[DataRecord] = {

val newRecord = new RichDataRecord(new DataRecord)

record

.getFeatureValue(UserStateBoundFeature)

.flatMap { userState => userStateToFeatureMap.get(userState.value) }.foreach {

booleanFeature => newRecord.setFeatureValue[JBoolean](booleanFeature, true)

}

record.getFeatureValue(UserCountryCodeFeature).foreach { countryCodeFeatureValue =>

newRecord.setFeatureValue[JString](USER\_COUNTRY\_ID, countryCodeFeatureValue)

}

val isUserVerifiedUnion =

UserVerifiedFeaturesSet.exists(feature => record.getFeatureValue(feature).getOrElse(false))

newRecord.setFeatureValue[JBoolean](UserHealthFeatures.IsUserVerifiedUnion, isUserVerifiedUnion)

List(newRecord.getRecord).asJava

}

}