package com.twitter.timelines.prediction.features.socialproof

import com.twitter.ml.api.DataRecord

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

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

import com.twitter.ml.api.Feature.SparseBinary

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

import com.twitter.timelines.prediction.features.socialproof.SocialProofDataRecordFeatures.\_

import com.twitter.timelines.socialproof.thriftscala.SocialProof

import com.twitter.timelines.socialproof.v1.thriftscala.SocialProofType

import com.twitter.timelines.util.CommonTypes.UserId

import scala.collection.JavaConverters.\_

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

abstract class SocialProofUserGroundTruth(userIds: Seq[UserId], count: Int) {

require(

count >= userIds.size,

"count must be equal to or greater than the number of entries in userIds"

)

// Using Double as the return type to make it more convenient for these values to be used as

// ML feature values.

val displayedUserCount: Double = userIds.size.toDouble

val undisplayedUserCount: Double = count - userIds.size.toDouble

val totalCount: Double = count.toDouble

def featureDisplayedUsers: SparseBinary

def featureDisplayedUserCount: Continuous

def featureUndisplayedUserCount: Continuous

def featureTotalUserCount: Continuous

def setFeatures(rec: DataRecord): Unit = {

rec.setFeatureValue(featureDisplayedUsers, toStringSet(userIds))

rec.setFeatureValue(featureDisplayedUserCount, displayedUserCount)

rec.setFeatureValue(featureUndisplayedUserCount, undisplayedUserCount)

rec.setFeatureValue(featureTotalUserCount, totalCount)

}

protected def toStringSet(value: Seq[Long]): Set[String] = {

value.map(\_.toString).toSet

}

}

case class FavoritedBySocialProofUserGroundTruth(userIds: Seq[UserId] = Seq.empty, count: Int = 0)

extends SocialProofUserGroundTruth(userIds, count) {

override val featureDisplayedUsers = SocialProofDisplayedFavoritedByUsers

override val featureDisplayedUserCount = SocialProofDisplayedFavoritedByUserCount

override val featureUndisplayedUserCount = SocialProofUndisplayedFavoritedByUserCount

override val featureTotalUserCount = SocialProofTotalFavoritedByUserCount

}

case class RetweetedBySocialProofUserGroundTruth(userIds: Seq[UserId] = Seq.empty, count: Int = 0)

extends SocialProofUserGroundTruth(userIds, count) {

override val featureDisplayedUsers = SocialProofDisplayedRetweetedByUsers

override val featureDisplayedUserCount = SocialProofDisplayedRetweetedByUserCount

override val featureUndisplayedUserCount = SocialProofUndisplayedRetweetedByUserCount

override val featureTotalUserCount = SocialProofTotalRetweetedByUserCount

}

case class RepliedBySocialProofUserGroundTruth(userIds: Seq[UserId] = Seq.empty, count: Int = 0)

extends SocialProofUserGroundTruth(userIds, count) {

override val featureDisplayedUsers = SocialProofDisplayedRepliedByUsers

override val featureDisplayedUserCount = SocialProofDisplayedRepliedByUserCount

override val featureUndisplayedUserCount = SocialProofUndisplayedRepliedByUserCount

override val featureTotalUserCount = SocialProofTotalRepliedByUserCount

}

case class SocialProofFeatures(

hasSocialProof: Boolean,

favoritedBy: FavoritedBySocialProofUserGroundTruth = FavoritedBySocialProofUserGroundTruth(),

retweetedBy: RetweetedBySocialProofUserGroundTruth = RetweetedBySocialProofUserGroundTruth(),

repliedBy: RepliedBySocialProofUserGroundTruth = RepliedBySocialProofUserGroundTruth()) {

def setFeatures(dataRecord: DataRecord): Unit =

if (hasSocialProof) {

dataRecord.setFeatureValue(HasSocialProof, hasSocialProof)

favoritedBy.setFeatures(dataRecord)

retweetedBy.setFeatures(dataRecord)

repliedBy.setFeatures(dataRecord)

}

}

object SocialProofFeatures {

def apply(socialProofs: Seq[SocialProof]): SocialProofFeatures =

socialProofs.foldLeft(SocialProofFeatures(hasSocialProof = socialProofs.nonEmpty))(

(prevFeatures, socialProof) => {

val userIds = socialProof.v1.userIds

val count = socialProof.v1.count

socialProof.v1.socialProofType match {

case SocialProofType.FavoritedBy =>

prevFeatures.copy(favoritedBy = FavoritedBySocialProofUserGroundTruth(userIds, count))

case SocialProofType.RetweetedBy =>

prevFeatures.copy(retweetedBy = RetweetedBySocialProofUserGroundTruth(userIds, count))

case SocialProofType.RepliedBy =>

prevFeatures.copy(repliedBy = RepliedBySocialProofUserGroundTruth(userIds, count))

case \_ =>

prevFeatures // skip silently instead of breaking jobs, since this isn't used yet

}

})

}

object SocialProofDataRecordFeatures {

val HasSocialProof = new Binary("recap.social\_proof.has\_social\_proof")

val SocialProofDisplayedFavoritedByUsers = new SparseBinary(

"recap.social\_proof.list.displayed.favorited\_by",

Set(UserId, PublicLikes, PrivateLikes).asJava

)

val SocialProofDisplayedFavoritedByUserCount = new Continuous(

"recap.social\_proof.count.displayed.favorited\_by",

Set(CountOfPrivateLikes, CountOfPublicLikes).asJava

)

val SocialProofUndisplayedFavoritedByUserCount = new Continuous(

"recap.social\_proof.count.undisplayed.favorited\_by",

Set(CountOfPrivateLikes, CountOfPublicLikes).asJava

)

val SocialProofTotalFavoritedByUserCount = new Continuous(

"recap.social\_proof.count.total.favorited\_by",

Set(CountOfPrivateLikes, CountOfPublicLikes).asJava

)

val SocialProofDisplayedRetweetedByUsers = new SparseBinary(

"recap.social\_proof.list.displayed.retweeted\_by",

Set(UserId, PublicRetweets, PrivateRetweets).asJava

)

val SocialProofDisplayedRetweetedByUserCount = new Continuous(

"recap.social\_proof.count.displayed.retweeted\_by",

Set(CountOfPrivateRetweets, CountOfPublicRetweets).asJava

)

val SocialProofUndisplayedRetweetedByUserCount = new Continuous(

"recap.social\_proof.count.undisplayed.retweeted\_by",

Set(CountOfPrivateRetweets, CountOfPublicRetweets).asJava

)

val SocialProofTotalRetweetedByUserCount = new Continuous(

"recap.social\_proof.count.total.retweeted\_by",

Set(CountOfPrivateRetweets, CountOfPublicRetweets).asJava

)

val SocialProofDisplayedRepliedByUsers = new SparseBinary(

"recap.social\_proof.list.displayed.replied\_by",

Set(UserId, PublicReplies, PrivateReplies).asJava

)

val SocialProofDisplayedRepliedByUserCount = new Continuous(

"recap.social\_proof.count.displayed.replied\_by",

Set(CountOfPrivateReplies, CountOfPublicReplies).asJava

)

val SocialProofUndisplayedRepliedByUserCount = new Continuous(

"recap.social\_proof.count.undisplayed.replied\_by",

Set(CountOfPrivateReplies, CountOfPublicReplies).asJava

)

val SocialProofTotalRepliedByUserCount = new Continuous(

"recap.social\_proof.count.total.replied\_by",

Set(CountOfPrivateReplies, CountOfPublicReplies).asJava

)

val AllFeatures = Seq(

HasSocialProof,

SocialProofDisplayedFavoritedByUsers,

SocialProofDisplayedFavoritedByUserCount,

SocialProofUndisplayedFavoritedByUserCount,

SocialProofTotalFavoritedByUserCount,

SocialProofDisplayedRetweetedByUsers,

SocialProofDisplayedRetweetedByUserCount,

SocialProofUndisplayedRetweetedByUserCount,

SocialProofTotalRetweetedByUserCount,

SocialProofDisplayedRepliedByUsers,

SocialProofDisplayedRepliedByUserCount,

SocialProofUndisplayedRepliedByUserCount,

SocialProofTotalRepliedByUserCount

)

}