package com.twitter.timelines.prediction.features.engagement\_features

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

import com.twitter.logging.Logger

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.Feature

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

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

import com.twitter.timelines.data\_processing.ml\_util.transforms.OneToSomeTransform

import com.twitter.timelines.data\_processing.ml\_util.transforms.RichITransform

import com.twitter.timelines.data\_processing.ml\_util.transforms.SparseBinaryUnion

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.TypedAggregateGroup

import com.twitter.timelineservice.suggests.features.engagement\_features.thriftscala.{

EngagementFeatures => ThriftEngagementFeatures

}

import com.twitter.timelineservice.suggests.features.engagement\_features.v1.thriftscala.{

EngagementFeatures => ThriftEngagementFeaturesV1

}

import scala.collection.JavaConverters.\_

object EngagementFeatures {

private[this] val logger = Logger.get(getClass.getSimpleName)

sealed trait EngagementFeature

case object Count extends EngagementFeature

case object RealGraphWeightAverage extends EngagementFeature

case object RealGraphWeightMax extends EngagementFeature

case object RealGraphWeightMin extends EngagementFeature

case object RealGraphWeightMissing extends EngagementFeature

case object RealGraphWeightVariance extends EngagementFeature

case object UserIds extends EngagementFeature

def fromThrift(thriftEngagementFeatures: ThriftEngagementFeatures): Option[EngagementFeatures] = {

thriftEngagementFeatures match {

case thriftEngagementFeaturesV1: ThriftEngagementFeatures.V1 =>

Some(

EngagementFeatures(

favoritedBy = thriftEngagementFeaturesV1.v1.favoritedBy,

retweetedBy = thriftEngagementFeaturesV1.v1.retweetedBy,

repliedBy = thriftEngagementFeaturesV1.v1.repliedBy,

)

)

case \_ => {

logger.error("Unexpected EngagementFeatures version found.")

None

}

}

}

val empty: EngagementFeatures = EngagementFeatures()

}

/\*\*

\* Contains user IDs who have engaged with a target entity, such as a Tweet,

\* and any additional data needed for derived features.

\*/

case class EngagementFeatures(

favoritedBy: Seq[Long] = Nil,

retweetedBy: Seq[Long] = Nil,

repliedBy: Seq[Long] = Nil,

realGraphWeightByUser: Map[Long, Double] = Map.empty) {

def isEmpty: Boolean = favoritedBy.isEmpty && retweetedBy.isEmpty && repliedBy.isEmpty

def nonEmpty: Boolean = !isEmpty

def toLogThrift: ThriftEngagementFeatures.V1 =

ThriftEngagementFeatures.V1(

ThriftEngagementFeaturesV1(

favoritedBy = favoritedBy,

retweetedBy = retweetedBy,

repliedBy = repliedBy

)

)

}

/\*\*

\* Represents engagement features derived from the Real Graph weight.

\*

\* These features are from the perspective of the source user, who is viewing their

\* timeline, to the destination users (or user), who created engagements.

\*

\* @param count number of engagements present

\* @param max max score of the engaging users

\* @param mean average score of the engaging users

\* @param min minimum score of the engaging users

\* @param missing for engagements present, how many Real Graph scores were missing

\* @param variance variance of scores of the engaging users

\*/

case class RealGraphDerivedEngagementFeatures(

count: Int,

max: Double,

mean: Double,

min: Double,

missing: Int,

variance: Double)

object EngagementDataRecordFeatures {

import EngagementFeatures.\_

val FavoritedByUserIds = new SparseBinary(

"engagement\_features.user\_ids.favorited\_by",

Set(UserId, PrivateLikes, PublicLikes).asJava)

val RetweetedByUserIds = new SparseBinary(

"engagement\_features.user\_ids.retweeted\_by",

Set(UserId, PrivateRetweets, PublicRetweets).asJava)

val RepliedByUserIds = new SparseBinary(

"engagement\_features.user\_ids.replied\_by",

Set(UserId, PrivateReplies, PublicReplies).asJava)

val InNetworkFavoritesCount = new Continuous(

"engagement\_features.in\_network.favorites.count",

Set(CountOfPrivateLikes, CountOfPublicLikes).asJava)

val InNetworkRetweetsCount = new Continuous(

"engagement\_features.in\_network.retweets.count",

Set(CountOfPrivateRetweets, CountOfPublicRetweets).asJava)

val InNetworkRepliesCount = new Continuous(

"engagement\_features.in\_network.replies.count",

Set(CountOfPrivateReplies, CountOfPublicReplies).asJava)

// real graph derived features

val InNetworkFavoritesAvgRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.favorites.avg\_weight",

Set(CountOfPrivateLikes, CountOfPublicLikes).asJava

)

val InNetworkFavoritesMaxRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.favorites.max\_weight",

Set(CountOfPrivateLikes, CountOfPublicLikes).asJava

)

val InNetworkFavoritesMinRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.favorites.min\_weight",

Set(CountOfPrivateLikes, CountOfPublicLikes).asJava

)

val InNetworkFavoritesRealGraphWeightMissing = new Continuous(

"engagement\_features.real\_graph.favorites.missing"

)

val InNetworkFavoritesRealGraphWeightVariance = new Continuous(

"engagement\_features.real\_graph.favorites.weight\_variance"

)

val InNetworkRetweetsMaxRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.retweets.max\_weight",

Set(CountOfPrivateRetweets, CountOfPublicRetweets).asJava

)

val InNetworkRetweetsMinRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.retweets.min\_weight",

Set(CountOfPrivateRetweets, CountOfPublicRetweets).asJava

)

val InNetworkRetweetsAvgRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.retweets.avg\_weight",

Set(CountOfPrivateRetweets, CountOfPublicRetweets).asJava

)

val InNetworkRetweetsRealGraphWeightMissing = new Continuous(

"engagement\_features.real\_graph.retweets.missing"

)

val InNetworkRetweetsRealGraphWeightVariance = new Continuous(

"engagement\_features.real\_graph.retweets.weight\_variance"

)

val InNetworkRepliesMaxRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.replies.max\_weight",

Set(CountOfPrivateReplies, CountOfPublicReplies).asJava

)

val InNetworkRepliesMinRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.replies.min\_weight",

Set(CountOfPrivateReplies, CountOfPublicReplies).asJava

)

val InNetworkRepliesAvgRealGraphWeight = new Continuous(

"engagement\_features.real\_graph.replies.avg\_weight",

Set(CountOfPrivateReplies, CountOfPublicReplies).asJava

)

val InNetworkRepliesRealGraphWeightMissing = new Continuous(

"engagement\_features.real\_graph.replies.missing"

)

val InNetworkRepliesRealGraphWeightVariance = new Continuous(

"engagement\_features.real\_graph.replies.weight\_variance"

)

sealed trait FeatureGroup {

def continuousFeatures: Map[EngagementFeature, Continuous]

def sparseBinaryFeatures: Map[EngagementFeature, SparseBinary]

def allFeatures: Seq[Feature[\_]] =

(continuousFeatures.values ++ sparseBinaryFeatures.values).toSeq

}

case object Favorites extends FeatureGroup {

override val continuousFeatures: Map[EngagementFeature, Continuous] =

Map(

Count -> InNetworkFavoritesCount,

RealGraphWeightAverage -> InNetworkFavoritesAvgRealGraphWeight,

RealGraphWeightMax -> InNetworkFavoritesMaxRealGraphWeight,

RealGraphWeightMin -> InNetworkFavoritesMinRealGraphWeight,

RealGraphWeightMissing -> InNetworkFavoritesRealGraphWeightMissing,

RealGraphWeightVariance -> InNetworkFavoritesRealGraphWeightVariance

)

override val sparseBinaryFeatures: Map[EngagementFeature, SparseBinary] =

Map(UserIds -> FavoritedByUserIds)

}

case object Retweets extends FeatureGroup {

override val continuousFeatures: Map[EngagementFeature, Continuous] =

Map(

Count -> InNetworkRetweetsCount,

RealGraphWeightAverage -> InNetworkRetweetsAvgRealGraphWeight,

RealGraphWeightMax -> InNetworkRetweetsMaxRealGraphWeight,

RealGraphWeightMin -> InNetworkRetweetsMinRealGraphWeight,

RealGraphWeightMissing -> InNetworkRetweetsRealGraphWeightMissing,

RealGraphWeightVariance -> InNetworkRetweetsRealGraphWeightVariance

)

override val sparseBinaryFeatures: Map[EngagementFeature, SparseBinary] =

Map(UserIds -> RetweetedByUserIds)

}

case object Replies extends FeatureGroup {

override val continuousFeatures: Map[EngagementFeature, Continuous] =

Map(

Count -> InNetworkRepliesCount,

RealGraphWeightAverage -> InNetworkRepliesAvgRealGraphWeight,

RealGraphWeightMax -> InNetworkRepliesMaxRealGraphWeight,

RealGraphWeightMin -> InNetworkRepliesMinRealGraphWeight,

RealGraphWeightMissing -> InNetworkRepliesRealGraphWeightMissing,

RealGraphWeightVariance -> InNetworkRepliesRealGraphWeightVariance

)

override val sparseBinaryFeatures: Map[EngagementFeature, SparseBinary] =

Map(UserIds -> RepliedByUserIds)

}

val PublicEngagerSets = Set(FavoritedByUserIds, RetweetedByUserIds, RepliedByUserIds)

val PublicEngagementUserIds = new SparseBinary(

"engagement\_features.user\_ids.public",

Set(UserId, EngagementsPublic).asJava

)

val ENGAGER\_ID = TypedAggregateGroup.sparseFeature(PublicEngagementUserIds)

val UnifyPublicEngagersTransform = SparseBinaryUnion(

featuresToUnify = PublicEngagerSets,

outputFeature = PublicEngagementUserIds

)

object RichUnifyPublicEngagersTransform extends OneToSomeTransform {

override def apply(dataRecord: DataRecord): Option[DataRecord] =

RichITransform(EngagementDataRecordFeatures.UnifyPublicEngagersTransform)(dataRecord)

override def featuresToTransform: Set[Feature[\_]] =

EngagementDataRecordFeatures.UnifyPublicEngagersTransform.featuresToUnify.toSet

}

}