package com.twitter.simclusters\_v2.scalding.embedding.abuse

import com.twitter.simclusters\_v2.common.ClusterId

import com.twitter.simclusters\_v2.thriftscala.{SimClusterWithScore, SimClustersEmbedding}

import com.twitter.util.Try

object ClusterPair {

def apply(

clusterId: ClusterId,

healthyScore: Double,

unhealthyScore: Double

): Option[ClusterPair] = {

if (healthyScore + unhealthyScore == 0.0) {

None

} else {

Some(new ClusterPair(clusterId, healthyScore, unhealthyScore))

}

}

}

case class ClusterPair private (

clusterId: ClusterId,

healthyScore: Double,

unhealthyScore: Double) {

def totalScores: Double = healthyScore + unhealthyScore

def healthRatio: Double = unhealthyScore / (unhealthyScore + healthyScore)

}

object PairedInteractionFeatures {

def smoothedHealthRatio(

unhealthySum: Double,

healthySum: Double,

smoothingFactor: Double,

prior: Double

): Double =

(unhealthySum + smoothingFactor \* prior) / (unhealthySum + healthySum + smoothingFactor)

}

/\*\*

\* Class used to derive features for abuse models. We pair a healthy embedding with an unhealthy

\* embedding. All the public methods on this class are derived features of these embeddings.

\*

\* @param healthyInteractionSimClusterEmbedding SimCluster embedding of healthy interactions (for

\* instance favs or impressions)

\* @param unhealthyInteractionSimClusterEmbedding SimCluster embedding of unhealthy interactions

\* (for instance blocks or abuse reports)

\*/

case class PairedInteractionFeatures(

healthyInteractionSimClusterEmbedding: SimClustersEmbedding,

unhealthyInteractionSimClusterEmbedding: SimClustersEmbedding) {

private[this] val scorePairs: Seq[ClusterPair] = {

val clusterToScoreMap = healthyInteractionSimClusterEmbedding.embedding.map {

simClusterWithScore =>

simClusterWithScore.clusterId -> simClusterWithScore.score

}.toMap

unhealthyInteractionSimClusterEmbedding.embedding.flatMap { simClusterWithScore =>

val clusterId = simClusterWithScore.clusterId

val postiveScoreOption = clusterToScoreMap.get(clusterId)

postiveScoreOption.flatMap { postiveScore =>

ClusterPair(clusterId, postiveScore, simClusterWithScore.score)

}

}

}

/\*\*

\* Get the pair of clusters with the most total interactions.

\*/

val highestScoreClusterPair: Option[ClusterPair] =

Try(scorePairs.maxBy(\_.totalScores)).toOption

/\*\*

\* Get the pair of clusters with the highest unhealthy to healthy ratio.

\*/

val highestHealthRatioClusterPair: Option[ClusterPair] =

Try(scorePairs.maxBy(\_.healthRatio)).toOption

/\*\*

\* Get the pair of clusters with the lowest unhealthy to healthy ratio.

\*/

val lowestHealthRatioClusterPair: Option[ClusterPair] =

Try(scorePairs.minBy(\_.healthRatio)).toOption

/\*\*

\* Get an embedding whose values are the ratio of unhealthy to healthy for that simcluster.

\*/

val healthRatioEmbedding: SimClustersEmbedding = {

val scores = scorePairs.map { pair =>

SimClusterWithScore(pair.clusterId, pair.healthRatio)

}

SimClustersEmbedding(scores)

}

/\*\*

\* Sum of the healthy scores for all the simclusters

\*/

val healthySum: Double = healthyInteractionSimClusterEmbedding.embedding.map(\_.score).sum

/\*\*

\* Sum of the unhealthy scores for all the simclusters

\*/

val unhealthySum: Double = unhealthyInteractionSimClusterEmbedding.embedding.map(\_.score).sum

/\*\*

\* ratio of unhealthy to healthy for all simclusters

\*/

val healthRatio: Double = unhealthySum / (unhealthySum + healthySum)

/\*\*

\* Ratio of unhealthy to healthy for all simclusters that is smoothed toward the prior with when

\* we have fewer observations.

\*

\* @param smoothingFactor The higher this value the more interactions we need to move the returned

\* ratio

\* @param prior The unhealthy to healthy for all interactions.

\*/

def smoothedHealthRatio(smoothingFactor: Double, prior: Double): Double =

PairedInteractionFeatures.smoothedHealthRatio(unhealthySum, healthySum, smoothingFactor, prior)

}