package com.twitter.follow\_recommendations.common.clients.real\_time\_real\_graph

import com.twitter.conversions.DurationOps.\_

import com.twitter.util.Time

object EngagementScorer {

private[real\_time\_real\_graph] val MemoryDecayHalfLife = 24.hour

private val ScoringFunctionBase = 0.5

def apply(

engagements: Map[Long, Seq[Engagement]],

engagementScoreMap: Map[EngagementType, Double],

minScore: Double = 0.0

): Seq[(Long, Double, Seq[EngagementType])] = {

val now = Time.now

engagements

.mapValues { engags =>

val totalScore = engags.map { engagement => score(engagement, now, engagementScoreMap) }.sum

val engagementProof = getEngagementProof(engags, engagementScoreMap)

(totalScore, engagementProof)

}

.collect { case (uid, (score, proof)) if score > minScore => (uid, score, proof) }

.toSeq

.sortBy(-\_.\_2)

}

/\*\*

\* The engagement score is the base score decayed via timestamp, loosely model the human memory forgetting

\* curve, see https://en.wikipedia.org/wiki/Forgetting\_curve

\*/

private[real\_time\_real\_graph] def score(

engagement: Engagement,

now: Time,

engagementScoreMap: Map[EngagementType, Double]

): Double = {

val timeLapse = math.max(now.inMillis - engagement.timestamp, 0)

val engagementScore = engagementScoreMap.getOrElse(engagement.engagementType, 0.0)

engagementScore \* math.pow(

ScoringFunctionBase,

timeLapse.toDouble / MemoryDecayHalfLife.inMillis)

}

private def getEngagementProof(

engagements: Seq[Engagement],

engagementScoreMap: Map[EngagementType, Double]

): Seq[EngagementType] = {

val filteredEngagement = engagements

.collectFirst {

case engagement

if engagement.engagementType != EngagementType.Click

&& engagementScoreMap.get(engagement.engagementType).exists(\_ > 0.0) =>

engagement.engagementType

}

Seq(filteredEngagement.getOrElse(EngagementType.Click))

}

}