package com.twitter.representationscorer.twistlyfeatures

import com.twitter.conversions.DurationOps.\_

import com.twitter.util.Duration

import com.twitter.util.Time

case class Engagements(

favs7d: Seq[UserSignal] = Nil,

retweets7d: Seq[UserSignal] = Nil,

follows30d: Seq[UserSignal] = Nil,

shares7d: Seq[UserSignal] = Nil,

replies7d: Seq[UserSignal] = Nil,

originalTweets7d: Seq[UserSignal] = Nil,

videoPlaybacks7d: Seq[UserSignal] = Nil,

block30d: Seq[UserSignal] = Nil,

mute30d: Seq[UserSignal] = Nil,

report30d: Seq[UserSignal] = Nil,

dontlike30d: Seq[UserSignal] = Nil,

seeFewer30d: Seq[UserSignal] = Nil) {

import Engagements.\_

private val now = Time.now

private val oneDayAgo = (now - OneDaySpan).inMillis

private val sevenDaysAgo = (now - SevenDaysSpan).inMillis

// All ids from the signals grouped by type (tweetIds, userIds, etc)

val tweetIds: Seq[Long] =

(favs7d ++ retweets7d ++ shares7d

++ replies7d ++ originalTweets7d ++ videoPlaybacks7d

++ report30d ++ dontlike30d ++ seeFewer30d)

.map(\_.targetId)

val authorIds: Seq[Long] = (follows30d ++ block30d ++ mute30d).map(\_.targetId)

// Tweet signals

val dontlike7d: Seq[UserSignal] = dontlike30d.filter(\_.timestamp > sevenDaysAgo)

val seeFewer7d: Seq[UserSignal] = seeFewer30d.filter(\_.timestamp > sevenDaysAgo)

val favs1d: Seq[UserSignal] = favs7d.filter(\_.timestamp > oneDayAgo)

val retweets1d: Seq[UserSignal] = retweets7d.filter(\_.timestamp > oneDayAgo)

val shares1d: Seq[UserSignal] = shares7d.filter(\_.timestamp > oneDayAgo)

val replies1d: Seq[UserSignal] = replies7d.filter(\_.timestamp > oneDayAgo)

val originalTweets1d: Seq[UserSignal] = originalTweets7d.filter(\_.timestamp > oneDayAgo)

val videoPlaybacks1d: Seq[UserSignal] = videoPlaybacks7d.filter(\_.timestamp > oneDayAgo)

val dontlike1d: Seq[UserSignal] = dontlike7d.filter(\_.timestamp > oneDayAgo)

val seeFewer1d: Seq[UserSignal] = seeFewer7d.filter(\_.timestamp > oneDayAgo)

// User signals

val follows7d: Seq[UserSignal] = follows30d.filter(\_.timestamp > sevenDaysAgo)

val block7d: Seq[UserSignal] = block30d.filter(\_.timestamp > sevenDaysAgo)

val mute7d: Seq[UserSignal] = mute30d.filter(\_.timestamp > sevenDaysAgo)

val report7d: Seq[UserSignal] = report30d.filter(\_.timestamp > sevenDaysAgo)

val block1d: Seq[UserSignal] = block7d.filter(\_.timestamp > oneDayAgo)

val mute1d: Seq[UserSignal] = mute7d.filter(\_.timestamp > oneDayAgo)

val report1d: Seq[UserSignal] = report7d.filter(\_.timestamp > oneDayAgo)

}

object Engagements {

val OneDaySpan: Duration = 1.days

val SevenDaysSpan: Duration = 7.days

val ThirtyDaysSpan: Duration = 30.days

}

case class UserSignal(targetId: Long, timestamp: Long)