package com.twitter.product\_mixer.component\_library.feature\_hydrator.candidate.decay

import com.twitter.conversions.DurationOps.\_

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.model.common.identifier.FeatureHydratorIdentifier

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.functional\_component.configapi.StaticParam

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.CandidateFeatureHydrator

import com.twitter.product\_mixer.core.model.common.UniversalNoun

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.Param

import com.twitter.util.Duration

object DecayScore extends Feature[UniversalNoun[Long], Double]

/\*\*

\* Hydrates snowflake ID candidates with a decay score:

\*

\* It is using exponential decay formula to calculate the score

\* exp(k \* age)

\* where k = ln(0.5) / half-life

\*

\* Here is an example for half-life = 1 day

\* For the brand new tweet it will be exp((ln(0.5)/1)\*0) = 1

\* For the tweet which was created 1 day ago it will be exp((ln(0.5)/1)\*1) = 0.5

\* For the tweet which was created 10 day ago it will be exp((ln(0.5)/1)\*10) = 0.00097

\*

\* Reference: https://www.cuemath.com/exponential-decay-formula/

\*

\* @note This penalizes but does not filter out the candidate, so "stale" candidates can still appear.

\*/

case class DecayCandidateFeatureHydrator[Candidate <: UniversalNoun[Long]](

halfLife: Param[Duration] = StaticParam[Duration](2.days),

resultFeature: Feature[UniversalNoun[Long], Double] = DecayScore)

extends CandidateFeatureHydrator[PipelineQuery, Candidate] {

override val features: Set[Feature[\_, \_]] = Set(resultFeature)

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("Decay")

override def apply(

query: PipelineQuery,

candidate: Candidate,

existingFeatures: FeatureMap

): Stitch[FeatureMap] = {

val halfLifeInMillis = query.params(halfLife).inMillis

val creationTime = SnowflakeId.timeFromId(candidate.id)

val ageInMillis = creationTime.untilNow.inMilliseconds

// it is using a exponential decay formula: e^(k \* tweetAge)

// where k = ln(0.5) / half-life

val k = math.log(0.5D) / halfLifeInMillis

val decayScore = math.exp(k \* ageInMillis)

Stitch.value(

FeatureMapBuilder()

.add(resultFeature, decayScore)

.build())

}

}