package com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.metrics

import java.lang.{Long => JLong}

import com.twitter.ml.api.\_

import com.twitter.ml.api.util.SRichDataRecord

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.metrics.ConversionUtils.\_

import com.twitter.util.Duration

import com.twitter.util.Time

import scala.math.max

/\*\*

\* This metric measures how recently an action has taken place. A value of 1.0

\* indicates the action happened just now. This value decays with time if the

\* action has not taken place and is reset to 1 when the action happens. So lower

\* value indicates a stale or older action.

\*

\* For example consider an action of "user liking a video". The last reset metric

\* value changes as follows for a half life of 1 day.

\*

\* ----------------------------------------------------------------------------

\* day | action | feature value | Description

\* ----------------------------------------------------------------------------

\* 1 | user likes the video | 1.0 | Set the value to 1

\* 2 | user does not like video | 0.5 | Decay the value

\* 3 | user does not like video | 0.25 | Decay the value

\* 4 | user likes the video | 1.0 | Reset the value to 1

\* -----------------------------------------------------------------------------

\*

\* @tparam T

\*/

case class TypedLastResetMetric[T]() extends TimedValueAggregationMetric[T] {

import AggregationMetricCommon.\_

override val operatorName = "last\_reset"

override def getIncrementValue(

record: DataRecord,

feature: Option[Feature[T]],

timestampFeature: Feature[JLong]

): TimedValue[Double] = {

val featureExists: Boolean = feature match {

case Some(f) => SRichDataRecord(record).hasFeature(f)

case None => true

}

TimedValue[Double](

value = booleanToDouble(featureExists),

timestamp = Time.fromMilliseconds(getTimestamp(record, timestampFeature))

)

}

private def getDecayedValue(

olderTimedValue: TimedValue[Double],

newerTimestamp: Time,

halfLife: Duration

): Double = {

if (halfLife.inMilliseconds == 0L) {

0.0

} else {

val timeDelta = newerTimestamp.inMilliseconds - olderTimedValue.timestamp.inMilliseconds

val resultValue = olderTimedValue.value / math.pow(2.0, timeDelta / halfLife.inMillis)

if (resultValue > AggregationMetricCommon.Epsilon) resultValue else 0.0

}

}

override def plus(

left: TimedValue[Double],

right: TimedValue[Double],

halfLife: Duration

): TimedValue[Double] = {

val (newerTimedValue, olderTimedValue) = if (left.timestamp > right.timestamp) {

(left, right)

} else {

(right, left)

}

val optionallyDecayedOlderValue = if (halfLife == Duration.Top) {

// Since we don't want to decay, older value is not changed

olderTimedValue.value

} else {

// Decay older value

getDecayedValue(olderTimedValue, newerTimedValue.timestamp, halfLife)

}

TimedValue[Double](

value = max(newerTimedValue.value, optionallyDecayedOlderValue),

timestamp = newerTimedValue.timestamp

)

}

override def zero(timeOpt: Option[Long]): TimedValue[Double] = TimedValue[Double](

value = 0.0,

timestamp = Time.fromMilliseconds(0)

)

}

/\*\*

\* Syntactic sugar for the last reset metric that works with

\* any feature type as opposed to being tied to a specific type.

\* See EasyMetric.scala for more details on why this is useful.

\*/

object LastResetMetric extends EasyMetric {

override def forFeatureType[T](

featureType: FeatureType

): Option[AggregationMetric[T, \_]] =

Some(TypedLastResetMetric[T]())

}