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

import com.twitter.ml.api.\_

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

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.metrics.AggregationMetricCommon.getTimestamp

import com.twitter.util.Duration

import com.twitter.util.Time

import java.lang.{Long => JLong}

import java.lang.{Number => JNumber}

import java.lang.{Double => JDouble}

import scala.math.max

case class TypedMaxMetric[T <: JNumber](defaultValue: Double = 0.0)

extends TimedValueAggregationMetric[T] {

override val operatorName = "max"

override def getIncrementValue(

dataRecord: DataRecord,

feature: Option[Feature[T]],

timestampFeature: Feature[JLong]

): TimedValue[Double] = {

val value = feature

.flatMap(SRichDataRecord(dataRecord).getFeatureValueOpt(\_))

.map(\_.doubleValue()).getOrElse(defaultValue)

val timestamp = Time.fromMilliseconds(getTimestamp(dataRecord, timestampFeature))

TimedValue[Double](value = value, timestamp = timestamp)

}

override def plus(

left: TimedValue[Double],

right: TimedValue[Double],

halfLife: Duration

): TimedValue[Double] = {

assert(

halfLife.toString == "Duration.Top",

s"halfLife must be Duration.Top when using max metric, but ${halfLife.toString} is used"

)

TimedValue[Double](

value = max(left.value, right.value),

timestamp = left.timestamp.max(right.timestamp)

)

}

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

TimedValue[Double](

value = 0.0,

timestamp = Time.fromMilliseconds(0)

)

}

object MaxMetric extends EasyMetric {

def forFeatureType[T](

featureType: FeatureType,

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

featureType match {

case FeatureType.CONTINUOUS =>

Some(TypedMaxMetric[JDouble]().asInstanceOf[AggregationMetric[T, Double]])

case FeatureType.DISCRETE =>

Some(TypedMaxMetric[JLong]().asInstanceOf[AggregationMetric[T, Double]])

case \_ => None

}

}