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

import com.twitter.ml.api.\_

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

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

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

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

import com.twitter.util.Duration

import com.twitter.util.Time

import java.lang.{Double => JDouble}

import java.lang.{Long => JLong}

import java.util.{Map => JMap}

/\*

\* ContinuousAggregationMetric overrides method AggregationMetric dealing

\* with reading and writing continuous values from a data record.

\*

\* operatorName is a string used for naming the resultant aggregate feature

\* (e.g. "count" if its a count feature, or "sum" if a sum feature).

\*/

trait TimedValueAggregationMetric[T] extends AggregationMetric[T, Double] {

import AggregationMetricCommon.\_

val operatorName: String

override def getAggregateValue(

record: DataRecord,

query: AggregateFeature[T],

aggregateOutputs: Option[List[JLong]] = None

): TimedValue[Double] = {

/\*

\* We know aggregateOutputs(0) will have the continuous feature,

\* since we put it there in getOutputFeatureIds() - see code below.

\* This helps us get a 4x speedup. Using any structure more complex

\* than a list was also a performance bottleneck.

\*/

val featureHash: JLong = aggregateOutputs

.getOrElse(getOutputFeatureIds(query))

.head

val continuousValueOption: Option[Double] = Option(record.continuousFeatures)

.flatMap { case jmap: JMap[JLong, JDouble] => Option(jmap.get(featureHash)) }

.map(\_.toDouble)

val timeOption = Option(record.discreteFeatures)

.flatMap { case jmap: JMap[JLong, JLong] => Option(jmap.get(TimestampHash)) }

.map(\_.toLong)

val resultOption: Option[TimedValue[Double]] = (continuousValueOption, timeOption) match {

case (Some(featureValue), Some(timesamp)) =>

Some(TimedValue[Double](featureValue, Time.fromMilliseconds(timesamp)))

case \_ => None

}

resultOption.getOrElse(zero(timeOption))

}

override def setAggregateValue(

record: DataRecord,

query: AggregateFeature[T],

aggregateOutputs: Option[List[JLong]] = None,

value: TimedValue[Double]

): Unit = {

/\*

\* We know aggregateOutputs(0) will have the continuous feature,

\* since we put it there in getOutputFeatureIds() - see code below.

\* This helps us get a 4x speedup. Using any structure more complex

\* than a list was also a performance bottleneck.

\*/

val featureHash: JLong = aggregateOutputs

.getOrElse(getOutputFeatureIds(query))

.head

/\* Only set value if non-zero to save space \*/

if (value.value != 0.0) {

record.putToContinuousFeatures(featureHash, value.value)

}

/\*

\* We do not set timestamp since that might affect correctness of

\* future aggregations due to the decay semantics.

\*/

}

/\* Only one feature stored in the aggregated datarecord: the result continuous value \*/

override def getOutputFeatures(query: AggregateFeature[T]): List[Feature[\_]] = {

val feature = cachedFullFeature(query, operatorName, FeatureType.CONTINUOUS)

List(feature)

}

}