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

import com.twitter.ml.api.\_

import com.twitter.ml.api.constant.SharedFeatures

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

import com.twitter.util.Duration

import java.lang.{Long => JLong}

/\*\*

\* Represents an aggregation operator (e.g. count or mean).

\* Override all functions in this trait to implement your own metric.

\* The operator is parameterized on an input type T, which is the type

\* of feature it aggregates, and a TimedValue[A] which is

\* the result type of aggregation for this metric.

\*/

trait AggregationMetric[T, A] extends FeatureCache[T] {

/\*

\* Combines two timed aggregate values ''left'' and ''right''

\* with the specified half life ''halfLife'' to produce a result

\* TimedValue

\*

\* @param left Left timed value

\* @param right Right timed value

\* @param halfLife Half life to use for adding timed values

\* @return Result timed value

\*/

def plus(left: TimedValue[A], right: TimedValue[A], halfLife: Duration): TimedValue[A]

/\*

\* Gets increment value given a datarecord and a feature.

\*

\* @param dataRecord to get increment value from.

\* @param feature Feature to get increment value for. If None,

then the semantics is to just aggregate the label.

\* @param timestampFeature Feature to use as millisecond timestamp

for decayed value aggregation.

\* @return The incremental contribution to the aggregate of ''feature'' from ''dataRecord''.

\*

\* For example, if the aggregation metric is count, the incremental

\* contribution is always a TimedValue (1.0, time). If the aggregation metric

\* is mean, and the feature is a continuous feature (double), the incremental

\* contribution looks like a tuple (value, 1.0, time)

\*/

def getIncrementValue(

dataRecord: DataRecord,

feature: Option[Feature[T]],

timestampFeature: Feature[JLong]

): TimedValue[A]

/\*

\* The "zero" value for aggregation.

\* For example, the zero is 0 for the count operator.

\*/

def zero(timeOpt: Option[Long] = None): TimedValue[A]

/\*

\* Gets the value of aggregate feature(s) stored in a datarecord, if any.

\* Different aggregate operators might store this info in the datarecord

\* differently. E.g. count just stores a count, while mean needs to

\* store both a sum and a count, and compile them into a TimedValue. We call

\* these features stored in the record "output" features.

\*

\* @param record Record to get value from

\* @param query AggregateFeature (see above) specifying details of aggregate

\* @param aggregateOutputs An optional precomputed set of aggregation "output"

\* feature hashes for this (query, metric) pair. This can be derived from ''query'',

\* but we precompute and pass this in for significantly (approximately 4x = 400%)

\* faster performance. If not passed in, the operator should reconstruct these features

\* from scratch.

\*

\* @return The aggregate value if found in ''record'', else the appropriate "zero"

for this type of aggregation.

\*/

def getAggregateValue(

record: DataRecord,

query: AggregateFeature[T],

aggregateOutputs: Option[List[JLong]] = None

): TimedValue[A]

/\*

\* Sets the value of aggregate feature(s) in a datarecord. Different operators

\* will have different representations (see example above).

\*

\* @param record Record to set value in

\* @param query AggregateFeature (see above) specifying details of aggregate

\* @param aggregateOutputs An optional precomputed set of aggregation "output"

\* features for this (query, metric) pair. This can be derived from ''query'',

\* but we precompute and pass this in for significantly (approximately 4x = 400%)

\* faster performance. If not passed in, the operator should reconstruct these features

\* from scratch.

\*

\* @param value Value to set for aggregate feature in the record being passed in via ''query''

\*/

def setAggregateValue(

record: DataRecord,

query: AggregateFeature[T],

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

value: TimedValue[A]

): Unit

/\*\*

\* Get features used to store aggregate output representation

\* in partially aggregated data records.

\*

\* @query AggregateFeature (see above) specifying details of aggregate

\* @return A list of "output" features used by this metric to store

\* output representation. For example, for the "count" operator, we

\* have only one element in this list, which is the result "count" feature.

\* For the "mean" operator, we have three elements in this list: the "count"

\* feature, the "sum" feature and the "mean" feature.

\*/

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

/\*\*

\* Get feature hashes used to store aggregate output representation

\* in partially aggregated data records.

\*

\* @query AggregateFeature (see above) specifying details of aggregate

\* @return A list of "output" feature hashes used by this metric to store

\* output representation. For example, for the "count" operator, we

\* have only one element in this list, which is the result "count" feature.

\* For the "mean" operator, we have three elements in this list: the "count"

\* feature, the "sum" feature and the "mean" feature.

\*/

def getOutputFeatureIds(query: AggregateFeature[T]): List[JLong] =

getOutputFeatures(query)

.map(\_.getDenseFeatureId().asInstanceOf[JLong])

/\*

\* Sums the given feature in two datarecords into a result record

\* WARNING: this method has side-effects; it modifies combined

\*

\* @param combined Result datarecord to mutate and store addition result in

\* @param left Left datarecord to add

\* @param right Right datarecord to add

\* @param query Details of aggregate to add

\* @param aggregateOutputs An optional precomputed set of aggregation "output"

\* feature hashes for this (query, metric) pair. This can be derived from ''query'',

\* but we precompute and pass this in for significantly (approximately 4x = 400%)

\* faster performance. If not passed in, the operator should reconstruct these features

\* from scratch.

\*/

def mutatePlus(

combined: DataRecord,

left: DataRecord,

right: DataRecord,

query: AggregateFeature[T],

aggregateOutputs: Option[List[JLong]] = None

): Unit = {

val leftValue = getAggregateValue(left, query, aggregateOutputs)

val rightValue = getAggregateValue(right, query, aggregateOutputs)

val combinedValue = plus(leftValue, rightValue, query.halfLife)

setAggregateValue(combined, query, aggregateOutputs, combinedValue)

}

/\*\*

\* Helper function to get increment value from an input DataRecord

\* and copy it to an output DataRecord, given an AggregateFeature query spec.

\*

\* @param output Datarecord to output increment to (will be mutated by this method)

\* @param input Datarecord to get increment from

\* @param query Details of aggregation

\* @param aggregateOutputs An optional precomputed set of aggregation "output"

\* feature hashes for this (query, metric) pair. This can be derived from ''query'',

\* but we precompute and pass this in for significantly (approximately 4x = 400%)

\* faster performance. If not passed in, the operator should reconstruct these features

\* from scratch.

\* @return True if an increment was set in the output record, else false

\*/

def setIncrement(

output: DataRecord,

input: DataRecord,

query: AggregateFeature[T],

timestampFeature: Feature[JLong] = SharedFeatures.TIMESTAMP,

aggregateOutputs: Option[List[JLong]] = None

): Boolean = {

if (query.label == None ||

(query.label.isDefined && SRichDataRecord(input).hasFeature(query.label.get))) {

val incrementValue: TimedValue[A] = getIncrementValue(input, query.feature, timestampFeature)

setAggregateValue(output, query, aggregateOutputs, incrementValue)

true

} else false

}

}