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

import com.twitter.ml.api.\_

import scala.collection.mutable

trait FeatureCache[T] {

/\*

\* Constructs feature names from scratch given an aggregate query and an output

\* feature name. E.g. given mean operator and "sum". This function is slow and should

\* only be called at pre-computation time.

\*

\* @param query Details of aggregate feature

\* @name Name of "output" feature for which we want to construct feature name

\* @return Full name of output feature

\*/

private def uncachedFullFeatureName(query: AggregateFeature[T], name: String): String =

List(query.featurePrefix, name).mkString(".")

/\*

\* A cache from (aggregate query, output feature name) -> fully qualified feature name

\* lazy since it doesn't need to be serialized to the mappers

\*/

private lazy val featureNameCache = mutable.Map[(AggregateFeature[T], String), String]()

/\*

\* A cache from (aggregate query, output feature name) -> precomputed output feature

\* lazy since it doesn't need to be serialized to the mappers

\*/

private lazy val featureCache = mutable.Map[(AggregateFeature[T], String), Feature[\_]]()

/\*\*

\* Given an (aggregate query, output feature name, output feature type),

\* look it up using featureNameCache and featureCache, falling back to uncachedFullFeatureName()

\* as a last resort to construct a precomputed output feature. Should only be

\* called at pre-computation time.

\*

\* @param query Details of aggregate feature

\* @name Name of "output" feature we want to precompute

\* @aggregateFeatureType type of "output" feature we want to precompute

\*/

def cachedFullFeature(

query: AggregateFeature[T],

name: String,

aggregateFeatureType: FeatureType

): Feature[\_] = {

lazy val cachedFeatureName = featureNameCache.getOrElseUpdate(

(query, name),

uncachedFullFeatureName(query, name)

)

def uncachedFullFeature(): Feature[\_] = {

val personalDataTypes =

AggregationMetricCommon.derivePersonalDataTypes(query.feature, query.label)

aggregateFeatureType match {

case FeatureType.BINARY => new Feature.Binary(cachedFeatureName, personalDataTypes)

case FeatureType.DISCRETE => new Feature.Discrete(cachedFeatureName, personalDataTypes)

case FeatureType.STRING => new Feature.Text(cachedFeatureName, personalDataTypes)

case FeatureType.CONTINUOUS => new Feature.Continuous(cachedFeatureName, personalDataTypes)

case FeatureType.SPARSE\_BINARY =>

new Feature.SparseBinary(cachedFeatureName, personalDataTypes)

case FeatureType.SPARSE\_CONTINUOUS =>

new Feature.SparseContinuous(cachedFeatureName, personalDataTypes)

}

}

featureCache.getOrElseUpdate(

(query, name),

uncachedFullFeature()

)

}

}