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

import com.twitter.dal.personal\_data.thriftjava.PersonalDataType

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.Feature

import com.twitter.ml.api.FeatureBuilder

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.thriftscala.{DataRecord => ScalaDataRecord}

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

import java.lang.{Double => JDouble}

import java.lang.{Long => JLong}

import scala.collection.JavaConverters.\_

/\*\*

\* Provides methods to build "scoped" aggregates, where base features generated by aggregates

\* V2 are scoped with a specific key.

\*

\* The class provides methods that take a Map of T -> DataRecord, where T is a key type, and

\* the DataRecord contains features produced by the aggregation\_framework. The methods then

\* generate a \_new\_ DataRecord, containing "scoped" aggregate features, where each scoped

\* feature has the value of the scope key in the feature name, and the value of the feature

\* is the value of the original aggregate feature in the corresponding value from the original

\* Map.

\*

\* For efficiency reasons, the builder is initialized with the set of features that should be

\* scoped and the set of keys for which scoping should be supported.

\*

\* To understand how scope feature names are constructed, consider the following:

\*

\* {{{

\* val features = Set(

\* new Feature.Continuous("user\_injection\_aggregate.pair.any\_label.any\_feature.5.days.count"),

\* new Feature.Continuous("user\_injection\_aggregate.pair.any\_label.any\_feature.10.days.count")

\* )

\* val scopes = Set(SuggestType.Recap, SuggestType.WhoToFollow)

\* val scopeName = "InjectionType"

\* val scopedAggregateBuilder = ScopedAggregateBuilder(features, scopes, scopeName)

\*

\* }}}

\*

\* Then, generated scoped features would be among the following:

\* - user\_injection\_aggregate.scoped.pair.any\_label.any\_feature.5.days.count/scope\_name=InjectionType/scope=Recap

\* - user\_injection\_aggregate.scoped.pair.any\_label.any\_feature.5.days.count/scope\_name=InjectionType/scope=WhoToFollow

\* - user\_injection\_aggregate.scoped.pair.any\_label.any\_feature.10.days.count/scope\_name=InjectionType/scope=Recap

\* - user\_injection\_aggregate.scoped.pair.any\_label.any\_feature.10.days.count/scope\_name=InjectionType/scope=WhoToFollow

\*

\* @param featuresToScope the set of features for which one should generate scoped versions

\* @param scopeKeys the set of scope keys to generate scopes with

\* @param scopeName a string indicating what the scopes represent. This is also added to the scoped feature

\* @tparam K the type of scope key

\*/

class ScopedAggregateBuilder[K](

featuresToScope: Set[Feature[JDouble]],

scopeKeys: Set[K],

scopeName: String) {

private[this] def buildScopedAggregateFeature(

baseName: String,

scopeValue: String,

personalDataTypes: java.util.Set[PersonalDataType]

): Feature[JDouble] = {

val components = baseName.split("\\.").toList

val newName = (components.head :: "scoped" :: components.tail).mkString(".")

new FeatureBuilder.Continuous()

.addExtensionDimensions("scope\_name", "scope")

.setBaseName(newName)

.setPersonalDataTypes(personalDataTypes)

.extensionBuilder()

.addExtension("scope\_name", scopeName)

.addExtension("scope", scopeValue)

.build()

}

/\*\*

\* Index of (base aggregate feature name, key) -> key scoped count feature.

\*/

private[this] val keyScopedAggregateMap: Map[(String, K), Feature[JDouble]] = {

featuresToScope.flatMap { feat =>

scopeKeys.map { key =>

(feat.getFeatureName, key) ->

buildScopedAggregateFeature(

feat.getFeatureName,

key.toString,

AggregationMetricCommon.derivePersonalDataTypes(Some(feat))

)

}

}.toMap

}

type ContinuousFeaturesMap = Map[JLong, JDouble]

/\*\*

\* Create key-scoped features for raw aggregate feature ID to value maps, partitioned by key.

\*/

private[this] def buildAggregates(featureMapsByKey: Map[K, ContinuousFeaturesMap]): DataRecord = {

val continuousFeatures = featureMapsByKey

.flatMap {

case (key, featureMap) =>

featuresToScope.flatMap { feature =>

val newFeatureOpt = keyScopedAggregateMap.get((feature.getFeatureName, key))

newFeatureOpt.flatMap { newFeature =>

featureMap.get(feature.getFeatureId).map(new JLong(newFeature.getFeatureId) -> \_)

}

}.toMap

}

new DataRecord().setContinuousFeatures(continuousFeatures.asJava)

}

/\*\*

\* Create key-scoped features for Java [[DataRecord]] aggregate records partitioned by key.

\*

\* As an example, if the provided Map includes the key `SuggestType.Recap`, and [[scopeKeys]]

\* includes this key, then for a feature "xyz.pair.any\_label.any\_feature.5.days.count", the method

\* will generate the scoped feature "xyz.scoped.pair.any\_label.any\_feature.5.days.count/scope\_name=InjectionType/scope=Recap",

\* with the value being the value of the original feature from the Map.

\*

\* @param aggregatesByKey a map from key to a continuous feature map (ie. feature ID -> Double)

\* @return a Java [[DataRecord]] containing key-scoped features

\*/

def buildAggregatesJava(aggregatesByKey: Map[K, DataRecord]): DataRecord = {

val featureMapsByKey = aggregatesByKey.mapValues(\_.continuousFeatures.asScala.toMap)

buildAggregates(featureMapsByKey)

}

/\*\*

\* Create key-scoped features for Scala [[DataRecord]] aggregate records partitioned by key.

\*

\* As an example, if the provided Map includes the key `SuggestType.Recap`, and [[scopeKeys]]

\* includes this key, then for a feature "xyz.pair.any\_label.any\_feature.5.days.count", the method

\* will generate the scoped feature "xyz.scoped.pair.any\_label.any\_feature.5.days.count/scope\_name=InjectionType/scope=Recap",

\* with the value being the value of the original feature from the Map.

\*

\* This is a convenience method for some use cases where aggregates are read from Scala

\* thrift objects. Note that this still returns a Java [[DataRecord]], since most ML API

\* use the Java version.

\*

\* @param aggregatesByKey a map from key to a continuous feature map (ie. feature ID -> Double)

\* @return a Java [[DataRecord]] containing key-scoped features

\*/

def buildAggregatesScala(aggregatesByKey: Map[K, ScalaDataRecord]): DataRecord = {

val featureMapsByKey =

aggregatesByKey

.mapValues { record =>

val featureMap = record.continuousFeatures.getOrElse(Map[Long, Double]()).toMap

featureMap.map { case (k, v) => new JLong(k) -> new JDouble(v) }

}

buildAggregates(featureMapsByKey)

}

/\*\*

\* Returns a [[FeatureContext]] including all possible scoped features generated using this builder.

\*

\* @return a [[FeatureContext]] containing all scoped features.

\*/

def scopedFeatureContext: FeatureContext = new FeatureContext(keyScopedAggregateMap.values.asJava)

}