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

import com.twitter.ml.api.\_

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

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

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

import com.twitter.timelines.data\_processing.ml\_util.transforms.OneToSomeTransform

import com.twitter.util.Duration

import java.lang.{Boolean => JBoolean}

import java.lang.{Long => JLong}

import scala.language.existentials

/\*\*

\* A wrapper for [[com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.TypedAggregateGroup]]

\* (see TypedAggregateGroup.scala) with some convenient syntactic sugar that avoids

\* the user having to specify different groups for different types of features.

\* Gets translated into multiple strongly typed TypedAggregateGroup(s)

\* by the buildTypedAggregateGroups() method defined below.

\*

\* @param inputSource Source to compute this aggregate over

\* @param preTransforms Sequence of [[ITransform]] that is applied to

\* data records pre-aggregation (e.g. discretization, renaming)

\* @param samplingTransformOpt Optional [[OneToSomeTransform]] that samples data record

\* @param aggregatePrefix Prefix to use for naming resultant aggregate features

\* @param keys Features to group by when computing the aggregates

\* (e.g. USER\_ID, AUTHOR\_ID). These must be either discrete, string or sparse binary.

\* Grouping by a sparse binary feature is different than grouping by a discrete or string

\* feature. For example, if you have a sparse binary feature WORDS\_IN\_TWEET which is

\* a set of all words in a tweet, then grouping by this feature generates a

\* separate aggregate mean/count/etc for each value of the feature (each word), and

\* not just a single aggregate count for different "sets of words"

\* @param features Features to aggregate (e.g. blender\_score or is\_photo).

\* @param labels Labels to cross the features with to make pair features, if any.

\* @param metrics Aggregation metrics to compute (e.g. count, mean)

\* @param halfLives Half lives to use for the aggregations, to be crossed with the above.

\* use Duration.Top for "forever" aggregations over an infinite time window (no decay).

\* @param outputStore Store to output this aggregate to

\* @param includeAnyFeature Aggregate label counts for any feature value

\* @param includeAnyLabel Aggregate feature counts for any label value (e.g. all impressions)

\* @param includeTimestampFeature compute max aggregate on timestamp feature

\* @param aggExclusionRegex Sequence of Regexes, which define features to

\*/

case class AggregateGroup(

inputSource: AggregateSource,

aggregatePrefix: String,

keys: Set[Feature[\_]],

features: Set[Feature[\_]],

labels: Set[\_ <: Feature[JBoolean]],

metrics: Set[EasyMetric],

halfLives: Set[Duration],

outputStore: AggregateStore,

preTransforms: Seq[OneToSomeTransform] = Seq.empty,

includeAnyFeature: Boolean = true,

includeAnyLabel: Boolean = true,

includeTimestampFeature: Boolean = false,

aggExclusionRegex: Seq[String] = Seq.empty) {

private def toStrongType[T](

metrics: Set[EasyMetric],

features: Set[Feature[\_]],

featureType: FeatureType

): TypedAggregateGroup[\_] = {

val underlyingMetrics: Set[AggregationMetric[T, \_]] =

metrics.flatMap(\_.forFeatureType[T](featureType))

val underlyingFeatures: Set[Feature[T]] = features

.map(\_.asInstanceOf[Feature[T]])

TypedAggregateGroup[T](

inputSource = inputSource,

aggregatePrefix = aggregatePrefix,

keysToAggregate = keys,

featuresToAggregate = underlyingFeatures,

labels = labels,

metrics = underlyingMetrics,

halfLives = halfLives,

outputStore = outputStore,

preTransforms = preTransforms,

includeAnyFeature,

includeAnyLabel,

aggExclusionRegex

)

}

private def timestampTypedAggregateGroup: TypedAggregateGroup[\_] = {

val metrics: Set[AggregationMetric[JLong, \_]] =

Set(MaxMetric.forFeatureType[JLong](TypedAggregateGroup.timestampFeature.getFeatureType).get)

TypedAggregateGroup[JLong](

inputSource = inputSource,

aggregatePrefix = aggregatePrefix,

keysToAggregate = keys,

featuresToAggregate = Set(TypedAggregateGroup.timestampFeature),

labels = Set.empty,

metrics = metrics,

halfLives = Set(Duration.Top),

outputStore = outputStore,

preTransforms = preTransforms,

includeAnyFeature = false,

includeAnyLabel = true,

aggExclusionRegex = Seq.empty

)

}

def buildTypedAggregateGroups(): List[TypedAggregateGroup[\_]] = {

val typedAggregateGroupsList = {

if (features.isEmpty) {

List(toStrongType(metrics, features, FeatureType.BINARY))

} else {

features

.groupBy(\_.getFeatureType())

.toList

.map {

case (featureType, features) =>

toStrongType(metrics, features, featureType)

}

}

}

val optionalTimestampTypedAggregateGroup =

if (includeTimestampFeature) List(timestampTypedAggregateGroup) else List()

typedAggregateGroupsList ++ optionalTimestampTypedAggregateGroup

}

}