package com.twitter.product\_mixer.core.feature.featuremap.datarecord

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.FeatureContext

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

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.feature.datarecord.\_

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.IllegalStateFailure

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

import scala.collection.JavaConverters.\_

/\*\*

\* Constructs a DataRecord from a FeatureMap, given a predefined set of features.

\*

\* @param features predefined set of BaseDataRecordFeatures that should be included in the output DataRecord.

\*/

class DataRecordExtractor[DRFeature <: BaseDataRecordFeature[\_, \_]](

features: Set[DRFeature]) {

private val featureContext = new FeatureContext(features.collect {

case dataRecordCompatible: DataRecordCompatible[\_] => dataRecordCompatible.mlFeature

}.asJava)

def fromDataRecord(dataRecord: DataRecord): FeatureMap = {

val featureMapBuilder = FeatureMapBuilder()

val richDataRecord = SRichDataRecord(dataRecord, featureContext)

features.foreach {

// FeatureStoreDataRecordFeature is currently not supported

case \_: FeatureStoreDataRecordFeature[\_, \_] =>

throw new UnsupportedOperationException(

"FeatureStoreDataRecordFeature cannot be extracted from a DataRecord")

case feature: DataRecordFeature[\_, \_] with DataRecordCompatible[\_] =>

// Java API will return null, so use Option to convert it to Scala Option which is None when null.

richDataRecord.getFeatureValueOpt(feature.mlFeature)(

feature.fromDataRecordFeatureValue) match {

case Some(value) =>

featureMapBuilder.add(feature.asInstanceOf[Feature[\_, feature.FeatureType]], value)

case None =>

featureMapBuilder.addFailure(

feature,

PipelineFailure(

IllegalStateFailure,

s"Required DataRecord feature is missing: ${feature.mlFeature.getFeatureName}")

)

}

case feature: DataRecordOptionalFeature[\_, \_] with DataRecordCompatible[\_] =>

val featureValue =

richDataRecord.getFeatureValueOpt(feature.mlFeature)(feature.fromDataRecordFeatureValue)

featureMapBuilder

.add(feature.asInstanceOf[Feature[\_, Option[feature.FeatureType]]], featureValue)

// DataRecordInAFeature is currently not supported

case \_: DataRecordInAFeature[\_] =>

throw new UnsupportedOperationException(

"DataRecordInAFeature cannot be extracted from a DataRecord")

}

featureMapBuilder.build()

}

}