package com.twitter.product\_mixer.component\_library.scorer.deepbird

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

import com.twitter.ml.prediction\_service.BatchPredictionRequest

import com.twitter.ml.prediction\_service.BatchPredictionResponse

import com.twitter.cortex.deepbird.thriftjava.{ModelSelector => TModelSelector}

import com.twitter.ml.api.DataRecord

import com.twitter.product\_mixer.component\_library.scorer.common.ModelSelector

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

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

import com.twitter.product\_mixer.core.feature.featuremap.datarecord.DataRecordConverter

import com.twitter.product\_mixer.core.feature.featuremap.datarecord.DataRecordExtractor

import com.twitter.product\_mixer.core.feature.featuremap.datarecord.FeaturesScope

import com.twitter.product\_mixer.core.functional\_component.scorer.Scorer

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.UniversalNoun

import com.twitter.product\_mixer.core.model.common.identifier.ScorerIdentifier

import scala.collection.JavaConverters.\_

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

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

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

import com.twitter.stitch.Stitch

import com.twitter.util.Future

abstract class BaseDeepbirdV2Scorer[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

QueryFeatures <: BaseDataRecordFeature[Query, \_],

CandidateFeatures <: BaseDataRecordFeature[Candidate, \_],

ResultFeatures <: BaseDataRecordFeature[Candidate, \_]

](

override val identifier: ScorerIdentifier,

modelIdSelector: ModelSelector[Query],

queryFeatures: FeaturesScope[QueryFeatures],

candidateFeatures: FeaturesScope[CandidateFeatures],

resultFeatures: Set[ResultFeatures])

extends Scorer[Query, Candidate] {

private val queryDataRecordConverter = new DataRecordConverter(queryFeatures)

private val candidateDataRecordConverter = new DataRecordConverter(candidateFeatures)

private val resultDataRecordExtractor = new DataRecordExtractor(resultFeatures)

require(resultFeatures.nonEmpty, "Result features cannot be empty")

override val features: Set[Feature[\_, \_]] = resultFeatures.asInstanceOf[Set[Feature[\_, \_]]]

def getBatchPredictions(

request: BatchPredictionRequest,

modelSelector: TModelSelector

): Future[BatchPredictionResponse]

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[Seq[FeatureMap]] = {

// Convert all candidate feature maps to java datarecords then to scala datarecords.

val thriftCandidateDataRecords = candidates.map { candidate =>

candidateDataRecordConverter.toDataRecord(candidate.features)

}

val request = new BatchPredictionRequest(thriftCandidateDataRecords.asJava)

// Convert the query feature map to data record if available.

query.features.foreach { featureMap =>

request.setCommonFeatures(queryDataRecordConverter.toDataRecord(featureMap))

}

val modelSelector = modelIdSelector

.apply(query).map { id =>

val selector = new TModelSelector()

selector.setId(id)

selector

}.orNull

Stitch.callFuture(getBatchPredictions(request, modelSelector)).map { response =>

val dataRecords = Option(response.predictions).map(\_.asScala).getOrElse(Seq.empty)

buildResults(candidates, dataRecords)

}

}

private def buildResults(

candidates: Seq[CandidateWithFeatures[Candidate]],

dataRecords: Seq[DataRecord]

): Seq[FeatureMap] = {

if (dataRecords.size != candidates.size) {

throw PipelineFailure(IllegalStateFailure, "Result Size mismatched candidates size")

}

dataRecords.map { resultDataRecord =>

resultDataRecordExtractor.fromDataRecord(resultDataRecord)

}

}

}