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

import com.twitter.ml.prediction.core.PredictionEngine

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

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

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

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

import com.twitter.stitch.Stitch

/\*\*

\* Scorer that locally loads a Deepbird model.

\* @param identifier Unique identifier for the scorer

\* @param predictionEngine Prediction Engine hosting the Deepbird model.

\* @param candidateFeatures The Candidate Features to convert and pass to the deepbird model.

\* @param resultFeatures The Candidate features returned by the model.

\* @tparam Query Type of pipeline query.

\* @tparam Candidate Type of candidates to score.

\* @tparam QueryFeatures type of the query level features consumed by the scorer.

\* @tparam CandidateFeatures type of the candidate level features consumed by the scorer.

\* @tparam ResultFeatures type of the candidate level features returned by the scorer.

\*/

class LollyPredictionEngineScorer[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

QueryFeatures <: BaseDataRecordFeature[Query, \_],

CandidateFeatures <: BaseDataRecordFeature[Candidate, \_],

ResultFeatures <: BaseDataRecordFeature[Candidate, \_]

](

override val identifier: ScorerIdentifier,

predictionEngine: PredictionEngine,

candidateFeatures: FeaturesScope[CandidateFeatures],

resultFeatures: Set[ResultFeatures])

extends Scorer[Query, Candidate] {

private val dataRecordAdapter = new DataRecordConverter(candidateFeatures)

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

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

private val resultsDataRecordExtractor = new DataRecordExtractor(resultFeatures)

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[Seq[FeatureMap]] = {

val featureMaps = candidates.map { candidateWithFeatures =>

val dataRecord = dataRecordAdapter.toDataRecord(candidateWithFeatures.features)

val predictionResponse = predictionEngine.apply(new PredictionRequest(dataRecord), true)

resultsDataRecordExtractor.fromDataRecord(predictionResponse.getPrediction)

}

Stitch.value(featureMaps)

}

}