package com.twitter.follow\_recommendations.common.candidate\_sources.stp

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.bijection.thrift.BinaryThriftCodec

import com.twitter.relevance.ep\_model.scorer.EPScorer

import com.twitter.relevance.ep\_model.scorer.ScorerUtil

import com.twitter.relevance.ep\_model.thrift

import com.twitter.relevance.ep\_model.thriftscala.EPScoringOptions

import com.twitter.relevance.ep\_model.thriftscala.EPScoringRequest

import com.twitter.relevance.ep\_model.thriftscala.EPScoringResponse

import com.twitter.relevance.ep\_model.thriftscala.Record

import com.twitter.stitch.Stitch

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

import scala.util.Success

case class ScoredResponse(score: Double, featuresBreakdown: Option[String] = None)

/\*\*

\* STP ML ranker trained using prehistoric ML framework

\*/

@Singleton

class EpStpScorer @Inject() (epScorer: EPScorer) {

private def getScore(responses: List[EPScoringResponse]): Option[ScoredResponse] =

responses.headOption

.flatMap { response =>

response.scores.flatMap {

\_.headOption.map(score => ScoredResponse(ScorerUtil.normalize(score)))

}

}

def getScoredResponse(

record: Record,

details: Boolean = false

): Stitch[Option[ScoredResponse]] = {

val scoringOptions = EPScoringOptions(

addFeaturesBreakDown = details,

addTransformerIntermediateRecords = details

)

val request = EPScoringRequest(auxFeatures = Some(Seq(record)), options = Some(scoringOptions))

Stitch.callFuture(

BinaryThriftCodec[thrift.EPScoringRequest]

.invert(BinaryScalaCodec(EPScoringRequest).apply(request))

.map { thriftRequest: thrift.EPScoringRequest =>

val responsesF = epScorer

.score(List(thriftRequest).asJava)

.map(

\_.asScala.toList

.map(response =>

BinaryScalaCodec(EPScoringResponse)

.invert(BinaryThriftCodec[thrift.EPScoringResponse].apply(response)))

.collect { case Success(response) => response }

)

responsesF.map(getScore)

}

.getOrElse(Future(None)))

}

}

object EpStpScorer {

val WithFeaturesBreakDown = false

}