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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.models.AccountProof

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.models.FollowProof

import com.twitter.follow\_recommendations.common.models.HasRecentFollowedUserIds

import com.twitter.follow\_recommendations.common.models.Reason

import com.twitter.onboarding.relevance.features.strongtie.{

StrongTieFeatures => StrongTieFeaturesWrapper

}

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import com.twitter.wtf.scalding.jobs.strong\_tie\_prediction.STPRecord

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class OnlineSTPSourceWithDeepbirdV2Scorer @Inject() (

dbv2StpScorer: Dbv2StpScorer,

stpGraphBuilder: STPGraphBuilder,

baseStatReceiver: StatsReceiver)

extends BaseOnlineSTPSource(stpGraphBuilder, baseStatReceiver) {

private val dbv2ScorerUsedCounter = statsReceiver.counter("dbv2\_scorer\_used")

private val dbv2ScorerFailureCounter = statsReceiver.counter("dbv2\_scorer\_failure")

private val dbv2ScorerSuccessCounter = statsReceiver.counter("dbv2\_scorer\_success")

override def getCandidates(

records: Seq[STPRecord],

request: HasClientContext with HasParams with HasRecentFollowedUserIds,

): Stitch[Seq[CandidateUser]] = {

val possibleCandidates: Seq[Stitch[Option[CandidateUser]]] = records.map { trainingRecord =>

dbv2ScorerUsedCounter.incr()

val score = dbv2StpScorer.getScoredResponse(trainingRecord)

score.map {

case None =>

dbv2ScorerFailureCounter.incr()

None

case Some(scoreVal) =>

dbv2ScorerSuccessCounter.incr()

Some(

CandidateUser(

id = trainingRecord.destinationId,

score = Some(OnlineSTPSourceWithDeepbirdV2Scorer.logitSubtraction(scoreVal)),

reason = Some(

Reason(Some(

AccountProof(followProof =

Some(FollowProof(trainingRecord.socialProof, trainingRecord.socialProof.size)))

)))

).withCandidateSourceAndFeatures(

identifier,

Seq(StrongTieFeaturesWrapper(trainingRecord.features)))

)

}

}

Stitch.collect(possibleCandidates).map { \_.flatten.sortBy(-\_.score.getOrElse(0.0)) }

}

}

object OnlineSTPSourceWithDeepbirdV2Scorer {

// The following two variables are the means for the distribution of scores coming from the legacy

// and DBv2 OnlineSTP models. We need this to calibrate the DBv2 scores and align the two means.

// BQ Link: https://console.cloud.google.com/bigquery?sq=213005704923:e06ac27e4db74385a77a4b538c531f82

private val legacyMeanScore = 0.0478208871192468

private val dbv2MeanScore = 0.238666097210261

// In below are the necessary functions to calibrate the scores such that the means are aligned.

private val EPS: Double = 1e-8

private val e: Double = math.exp(1)

private def sigmoid(x: Double): Double = math.pow(e, x) / (math.pow(e, x) + 1)

// We add an EPS to the denominator to avoid division by 0.

private def logit(x: Double): Double = math.log(x / (1 - x + EPS))

def logitSubtraction(x: Double): Double = sigmoid(

logit(x) - (logit(dbv2MeanScore) - logit(legacyMeanScore)))

}