package com.twitter.product\_mixer.component\_library.scorer.cr\_ml\_ranker

import com.twitter.cr\_ml\_ranker.{thriftscala => t}

import com.twitter.product\_mixer.component\_library.model.candidate.BaseTweetCandidate

import com.twitter.stitch.SeqGroup

import com.twitter.stitch.Stitch

import com.twitter.util.Future

import com.twitter.util.Return

import com.twitter.util.Try

case class CrMlRankerResult(

tweetId: Long,

score: Double)

class CrMlRankerScoreStitchClient(

crMLRanker: t.CrMLRanker.MethodPerEndpoint,

maxBatchSize: Int) {

def getScore(

userId: Long,

tweetCandidate: BaseTweetCandidate,

rankingConfig: t.RankingConfig,

commonFeatures: t.CommonFeatures

): Stitch[CrMlRankerResult] = {

Stitch.call(

tweetCandidate,

CrMlRankerGroup(

userId = userId,

rankingConfig = rankingConfig,

commonFeatures = commonFeatures

)

)

}

private case class CrMlRankerGroup(

userId: Long,

rankingConfig: t.RankingConfig,

commonFeatures: t.CommonFeatures)

extends SeqGroup[BaseTweetCandidate, CrMlRankerResult] {

override val maxSize: Int = maxBatchSize

override protected def run(

tweetCandidates: Seq[BaseTweetCandidate]

): Future[Seq[Try[CrMlRankerResult]]] = {

val crMlRankerCandidates =

tweetCandidates.map { tweetCandidate =>

t.RankingCandidate(

tweetId = tweetCandidate.id,

hydrationContext = Some(

t.FeatureHydrationContext.HomeHydrationContext(t

.HomeFeatureHydrationContext(tweetAuthor = None)))

)

}

val thriftResults = crMLRanker.getRankedResults(

t.RankingRequest(

requestContext = t.RankingRequestContext(

userId = userId,

config = rankingConfig

),

candidates = crMlRankerCandidates,

commonFeatures = commonFeatures.commonFeatures

)

)

thriftResults.map { response =>

response.scoredTweets.map { scoredTweet =>

Return(

CrMlRankerResult(

tweetId = scoredTweet.tweetId,

score = scoredTweet.score

)

)

}

}

}

}

}