package com.twitter.timelineranker.clients

import com.twitter.cortex\_core.thriftscala.ModelName

import com.twitter.cortex\_tweet\_annotate.thriftscala.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.logging.Logger

import com.twitter.mediaservices.commons.mediainformation.thriftscala.CalibrationLevel

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.util.stats.RequestScope

import com.twitter.timelines.util.stats.RequestStats

import com.twitter.timelines.util.stats.ScopedFactory

import com.twitter.timelines.util.FailOpenHandler

import com.twitter.util.Future

object CortexTweetQueryServiceClient {

private[this] val logger = Logger.get(getClass.getSimpleName)

/\*\*

\* A tweet is considered safe if Cortex NSFA model gives it a score that is above the threshold.

\* Both the score and the threshold are returned in a response from getTweetSignalByIds endpoint.

\*/

private def getSafeTweet(

request: TweetSignalRequest,

response: ModelResponseResult

): Option[TweetId] = {

val tweetId = request.tweetId

response match {

case ModelResponseResult(ModelResponseState.Success, Some(tid), Some(modelResponse), \_) =>

val prediction = modelResponse.predictions.flatMap(\_.headOption)

val score = prediction.map(\_.score.score)

val highRecallBucket = prediction.flatMap(\_.calibrationBuckets).flatMap { buckets =>

buckets.find(\_.description.contains(CalibrationLevel.HighRecall))

}

val threshold = highRecallBucket.map(\_.threshold)

(score, threshold) match {

case (Some(s), Some(t)) if (s > t) =>

Some(tid)

case (Some(s), Some(t)) =>

logger.ifDebug(

s"Cortex NSFA score for tweet $tweetId is $s (threshold is $t), removing as unsafe."

)

None

case \_ =>

logger.ifDebug(s"Unexpected response, removing tweet $tweetId as unsafe.")

None

}

case \_ =>

logger.ifWarning(

s"Cortex tweet NSFA call was not successful, removing tweet $tweetId as unsafe."

)

None

}

}

}

/\*\*

\* Enables calling cortex tweet query service to get NSFA scores on the tweet.

\*/

class CortexTweetQueryServiceClient(

cortexClient: CortexTweetQueryService.MethodPerEndpoint,

requestScope: RequestScope,

statsReceiver: StatsReceiver)

extends RequestStats {

import CortexTweetQueryServiceClient.\_

private[this] val logger = Logger.get(getClass.getSimpleName)

private[this] val getTweetSignalByIdsRequestStats =

requestScope.stats("cortex", statsReceiver, suffix = Some("getTweetSignalByIds"))

private[this] val getTweetSignalByIdsRequestScopedStatsReceiver =

getTweetSignalByIdsRequestStats.scopedStatsReceiver

private[this] val failedCortexTweetQueryServiceScope =

getTweetSignalByIdsRequestScopedStatsReceiver.scope(Failures)

private[this] val failedCortexTweetQueryServiceCallCounter =

failedCortexTweetQueryServiceScope.counter("failOpen")

private[this] val cortexTweetQueryServiceFailOpenHandler = new FailOpenHandler(

getTweetSignalByIdsRequestScopedStatsReceiver

)

def getSafeTweets(tweetIds: Seq[TweetId]): Future[Seq[TweetId]] = {

val requests = tweetIds.map { id => TweetSignalRequest(id, ModelName.TweetToNsfa) }

val results = cortexClient

.getTweetSignalByIds(

GetTweetSignalByIdsRequest(requests)

)

.map(\_.results)

cortexTweetQueryServiceFailOpenHandler(

results.map { responses =>

requests.zip(responses).flatMap {

case (request, response) =>

getSafeTweet(request, response)

}

}

) { \_ =>

failedCortexTweetQueryServiceCallCounter.incr()

logger.ifWarning(s"Cortex tweet NSFA call failed, considering tweets $tweetIds as unsafe.")

Future.value(Seq())

}

}

}

class ScopedCortexTweetQueryServiceClientFactory(

cortexClient: CortexTweetQueryService.MethodPerEndpoint,

statsReceiver: StatsReceiver)

extends ScopedFactory[CortexTweetQueryServiceClient] {

override def scope(scope: RequestScope): CortexTweetQueryServiceClient = {

new CortexTweetQueryServiceClient(cortexClient, scope, statsReceiver)

}

}