package com.twitter.frigate.pushservice.model.candidate

import com.twitter.frigate.pushservice.model.PushTypes.PushCandidate

import com.twitter.frigate.pushservice.params.HighQualityScribingScores

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.params.PushMLModel

import com.twitter.util.Future

import java.util.concurrent.ConcurrentHashMap

import scala.collection.concurrent.{Map => CMap}

import scala.collection.convert.decorateAsScala.\_

trait QualityScribing {

self: PushCandidate with MLScores =>

// Use to store other scores (to avoid duplicate queries to other services, e.g. HSS)

private val externalCachedScores: CMap[String, Future[Option[Double]]] =

new ConcurrentHashMap[String, Future[Option[Double]]]().asScala

/\*\*

\* Retrieves the model version as specified by the corresponding FS param.

\* This model version will be used for getting the cached score or triggering

\* a prediction request.

\*

\* @param modelName The score we will like to scribe

\*/

private def getModelVersion(

modelName: HighQualityScribingScores.Name

): String = {

modelName match {

case HighQualityScribingScores.HeavyRankingScore =>

target.params(PushFeatureSwitchParams.HighQualityCandidatesHeavyRankingModel)

case HighQualityScribingScores.NonPersonalizedQualityScoreUsingCnn =>

target.params(PushFeatureSwitchParams.HighQualityCandidatesNonPersonalizedQualityCnnModel)

case HighQualityScribingScores.BqmlNsfwScore =>

target.params(PushFeatureSwitchParams.HighQualityCandidatesBqmlNsfwModel)

case HighQualityScribingScores.BqmlReportScore =>

target.params(PushFeatureSwitchParams.HighQualityCandidatesBqmlReportModel)

}

}

/\*\*

\* Retrieves the score for scribing either from a cached value or

\* by generating a prediction request. This will increase model QPS

\*

\* @param pushMLModel This represents the prefix of the model name (i.e. [pushMLModel]\_[version])

\* @param scoreName The name to be use when scribing this score

\*/

def getScribingScore(

pushMLModel: PushMLModel.Value,

scoreName: HighQualityScribingScores.Name

): Future[(String, Option[Double])] = {

getMLModelScore(

pushMLModel,

getModelVersion(scoreName)

).map { scoreOpt =>

scoreName.toString -> scoreOpt

}

}

/\*\*

\* Retrieves the score for scribing if it has been computed/cached before otherwise

\* it will return Future.None

\*

\* @param pushMLModel This represents the prefix of the model name (i.e. [pushMLModel]\_[version])

\* @param scoreName The name to be use when scribing this score

\*/

def getScribingScoreWithoutUpdate(

pushMLModel: PushMLModel.Value,

scoreName: HighQualityScribingScores.Name

): Future[(String, Option[Double])] = {

getMLModelScoreWithoutUpdate(

pushMLModel,

getModelVersion(scoreName)

).map { scoreOpt =>

scoreName.toString -> scoreOpt

}

}

/\*\*

\* Caches the given score future

\*

\* @param scoreName The name to be use when scribing this score

\* @param scoreFut Future mapping scoreName -> scoreOpt

\*/

def cacheExternalScore(scoreName: String, scoreFut: Future[Option[Double]]) = {

if (!externalCachedScores.contains(scoreName)) {

externalCachedScores += scoreName -> scoreFut

}

}

/\*\*

\* Returns all external scores future cached as a sequence

\*/

def getExternalCachedScores: Seq[Future[(String, Option[Double])]] = {

externalCachedScores.map {

case (modelName, scoreFut) =>

scoreFut.map { scoreOpt => modelName -> scoreOpt }

}.toSeq

}

def getExternalCachedScoreByName(name: String): Future[Option[Double]] = {

externalCachedScores.getOrElse(name, Future.None)

}

}