package com.twitter.frigate.pushservice.ml

import com.twitter.cortex.deepbird.thriftjava.ModelSelector

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateDetails

import com.twitter.frigate.common.base.FeatureMap

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

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

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

import com.twitter.frigate.pushservice.params.PushModelName

import com.twitter.frigate.pushservice.params.WeightedOpenOrNtabClickModel

import com.twitter.nrel.heavyranker.PushCandidateHydrationContextWithModel

import com.twitter.nrel.heavyranker.PushPredictionServiceStore

import com.twitter.nrel.heavyranker.TargetFeatureMapWithModel

import com.twitter.timelines.configapi.FSParam

import com.twitter.util.Future

/\*\*

\* PushMLModelScorer scores the Candidates and populates their ML scores

\*

\* @param pushMLModel Enum to specify which model to use for scoring the Candidates

\* @param modelToPredictionServiceStoreMap Supports all other prediction services. Specifies model ID -> dbv2 ReadableStore

\* @param defaultDBv2PredictionServiceStore: Supports models that are not specified in the previous maps (which will be directly configured in the config repo)

\* @param scoringStats StatsReceiver for scoping stats

\*/

class PushMLModelScorer(

pushMLModel: PushMLModel.Value,

modelToPredictionServiceStoreMap: Map[

WeightedOpenOrNtabClickModel.ModelNameType,

PushPredictionServiceStore

],

defaultDBv2PredictionServiceStore: PushPredictionServiceStore,

scoringStats: StatsReceiver) {

val queriesOutsideTheModelMaps: StatsReceiver =

scoringStats.scope("queries\_outside\_the\_model\_maps")

val totalQueriesOutsideTheModelMaps: Counter =

queriesOutsideTheModelMaps.counter("total")

private def scoreByBatchPredictionForModelFromMultiModelService(

predictionServiceStore: PushPredictionServiceStore,

modelVersion: WeightedOpenOrNtabClickModel.ModelNameType,

candidatesDetails: Seq[CandidateDetails[PushCandidate]],

useCommonFeatures: Boolean,

overridePushMLModel: PushMLModel.Value

): Seq[CandidateDetails[PushCandidate]] = {

val modelName =

PushModelName(overridePushMLModel, modelVersion).toString

val modelSelector = new ModelSelector()

modelSelector.setId(modelName)

val candidateHydrationWithFeaturesMap = candidatesDetails.map { candidatesDetail =>

(

candidatesDetail.candidate.candidateHydrationContext,

candidatesDetail.candidate.candidateFeatureMap())

}

if (candidatesDetails.nonEmpty) {

val candidatesWithScore = predictionServiceStore.getBatchPredictionsForModel(

candidatesDetails.head.candidate.target.targetHydrationContext,

candidatesDetails.head.candidate.target.featureMap,

candidateHydrationWithFeaturesMap,

Some(modelSelector),

useCommonFeatures

)

candidatesDetails.zip(candidatesWithScore).foreach {

case (candidateDetail, (\_, scoreOptFut)) =>

candidateDetail.candidate.populateQualityModelScore(

overridePushMLModel,

modelVersion,

scoreOptFut

)

}

}

candidatesDetails

}

private def scoreByBatchPrediction(

modelVersion: WeightedOpenOrNtabClickModel.ModelNameType,

candidatesDetails: Seq[CandidateDetails[PushCandidate]],

useCommonFeaturesForDBv2Service: Boolean,

overridePushMLModel: PushMLModel.Value

): Seq[CandidateDetails[PushCandidate]] = {

if (modelToPredictionServiceStoreMap.contains(modelVersion)) {

scoreByBatchPredictionForModelFromMultiModelService(

modelToPredictionServiceStoreMap(modelVersion),

modelVersion,

candidatesDetails,

useCommonFeaturesForDBv2Service,

overridePushMLModel

)

} else {

totalQueriesOutsideTheModelMaps.incr()

queriesOutsideTheModelMaps.counter(modelVersion).incr()

scoreByBatchPredictionForModelFromMultiModelService(

defaultDBv2PredictionServiceStore,

modelVersion,

candidatesDetails,

useCommonFeaturesForDBv2Service,

overridePushMLModel

)

}

}

def scoreByBatchPredictionForModelVersion(

target: Target,

candidatesDetails: Seq[CandidateDetails[PushCandidate]],

modelVersionParam: FSParam[WeightedOpenOrNtabClickModel.ModelNameType],

useCommonFeaturesForDBv2Service: Boolean = true,

overridePushMLModelOpt: Option[PushMLModel.Value] = None

): Seq[CandidateDetails[PushCandidate]] = {

scoreByBatchPrediction(

target.params(modelVersionParam),

candidatesDetails,

useCommonFeaturesForDBv2Service,

overridePushMLModelOpt.getOrElse(pushMLModel)

)

}

def singlePredicationForModelVersion(

modelVersion: String,

candidate: PushCandidate,

overridePushMLModelOpt: Option[PushMLModel.Value] = None

): Future[Option[Double]] = {

val modelSelector = new ModelSelector()

modelSelector.setId(

PushModelName(overridePushMLModelOpt.getOrElse(pushMLModel), modelVersion).toString

)

if (modelToPredictionServiceStoreMap.contains(modelVersion)) {

modelToPredictionServiceStoreMap(modelVersion).get(

PushCandidateHydrationContextWithModel(

candidate.target.targetHydrationContext,

candidate.target.featureMap,

candidate.candidateHydrationContext,

candidate.candidateFeatureMap(),

Some(modelSelector)

)

)

} else {

totalQueriesOutsideTheModelMaps.incr()

queriesOutsideTheModelMaps.counter(modelVersion).incr()

defaultDBv2PredictionServiceStore.get(

PushCandidateHydrationContextWithModel(

candidate.target.targetHydrationContext,

candidate.target.featureMap,

candidate.candidateHydrationContext,

candidate.candidateFeatureMap(),

Some(modelSelector)

)

)

}

}

def singlePredictionForTargetLevel(

modelVersion: String,

targetId: Long,

featureMap: Future[FeatureMap]

): Future[Option[Double]] = {

val modelSelector = new ModelSelector()

modelSelector.setId(

PushModelName(pushMLModel, modelVersion).toString

)

defaultDBv2PredictionServiceStore.getForTargetLevel(

TargetFeatureMapWithModel(targetId, featureMap, Some(modelSelector))

)

}

def getScoreHistogramCounters(

stats: StatsReceiver,

scopeName: String,

histogramBinSize: Double

): IndexedSeq[Counter] = {

val histogramScopedStatsReceiver = stats.scope(scopeName)

val numBins = math.ceil(1.0 / histogramBinSize).toInt

(0 to numBins) map { k =>

if (k == 0)

histogramScopedStatsReceiver.counter("candidates\_with\_scores\_zero")

else {

val counterName = "candidates\_with\_scores\_from\_%s\_to\_%s".format(

"%.2f".format(histogramBinSize \* (k - 1)).replace(".", ""),

"%.2f".format(math.min(1.0, histogramBinSize \* k)).replace(".", ""))

histogramScopedStatsReceiver.counter(counterName)

}

}

}

}