package com.twitter.frigate.pushservice.rank

import com.twitter.contentrecommender.thriftscala.LightRankingCandidate

import com.twitter.contentrecommender.thriftscala.LightRankingFeatureHydrationContext

import com.twitter.contentrecommender.thriftscala.MagicRecsFeatureHydrationContext

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.frigate.common.base.RandomRanker

import com.twitter.frigate.common.base.Ranker

import com.twitter.frigate.common.base.TweetAuthor

import com.twitter.frigate.common.base.TweetCandidate

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

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

import com.twitter.frigate.pushservice.params.PushConstants

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

import com.twitter.frigate.pushservice.params.PushParams

import com.twitter.ml.featurestore.lib.UserId

import com.twitter.nrel.lightranker.MagicRecsServeDataRecordLightRanker

import com.twitter.util.Future

class RFPHLightRanker(

lightRanker: MagicRecsServeDataRecordLightRanker,

stats: StatsReceiver)

extends Ranker[Target, PushCandidate] {

private val statsReceiver = stats.scope(this.getClass.getSimpleName)

private val lightRankerCandidateCounter = statsReceiver.counter("light\_ranker\_candidate\_count")

private val lightRankerRequestCounter = statsReceiver.counter("light\_ranker\_request\_count")

private val lightRankingStats: StatsReceiver = statsReceiver.scope("light\_ranking")

private val restrictLightRankingCounter: Counter =

lightRankingStats.counter("restrict\_light\_ranking")

private val selectedLightRankerScribedTargetCandidateCountStats: Stat =

lightRankingStats.stat("selected\_light\_ranker\_scribed\_target\_candidate\_count")

private val selectedLightRankerScribedCandidatesStats: Stat =

lightRankingStats.stat("selected\_light\_ranker\_scribed\_candidates")

private val lightRankingRandomBaselineStats: StatsReceiver =

statsReceiver.scope("light\_ranking\_random\_baseline")

override def rank(

target: Target,

candidates: Seq[CandidateDetails[PushCandidate]]

): Future[Seq[CandidateDetails[PushCandidate]]] = {

val enableLightRanker = target.params(PushFeatureSwitchParams.EnableLightRankingParam)

val restrictLightRanker = target.params(PushParams.RestrictLightRankingParam)

val lightRankerSelectionThreshold =

target.params(PushFeatureSwitchParams.LightRankingNumberOfCandidatesParam)

val randomRanker = RandomRanker[Target, PushCandidate]()(lightRankingRandomBaselineStats)

if (enableLightRanker && candidates.length > lightRankerSelectionThreshold && !target.scribeFeatureForRequestScribe) {

val (tweetCandidates, nonTweetCandidates) =

candidates.partition {

case CandidateDetails(pushCandidate: PushCandidate with TweetCandidate, source) => true

case \_ => false

}

val lightRankerSelectedTweetCandidatesFut = {

if (restrictLightRanker) {

restrictLightRankingCounter.incr()

lightRankThenTake(

target,

tweetCandidates

.asInstanceOf[Seq[CandidateDetails[PushCandidate with TweetCandidate]]],

PushConstants.RestrictLightRankingCandidatesThreshold

)

} else if (target.params(PushFeatureSwitchParams.EnableRandomBaselineLightRankingParam)) {

randomRanker.rank(target, tweetCandidates).map { randomLightRankerCands =>

randomLightRankerCands.take(lightRankerSelectionThreshold)

}

} else {

lightRankThenTake(

target,

tweetCandidates

.asInstanceOf[Seq[CandidateDetails[PushCandidate with TweetCandidate]]],

lightRankerSelectionThreshold

)

}

}

lightRankerSelectedTweetCandidatesFut.map { returnedTweetCandidates =>

nonTweetCandidates ++ returnedTweetCandidates

}

} else if (target.scribeFeatureForRequestScribe) {

val downSampleRate: Double =

if (target.params(PushParams.DownSampleLightRankingScribeCandidatesParam))

PushConstants.DownSampleLightRankingScribeCandidatesRate

else target.params(PushFeatureSwitchParams.LightRankingScribeCandidatesDownSamplingParam)

val selectedCandidateCounter: Int = math.ceil(candidates.size \* downSampleRate).toInt

selectedLightRankerScribedTargetCandidateCountStats.add(selectedCandidateCounter.toFloat)

randomRanker.rank(target, candidates).map { randomLightRankerCands =>

val selectedCandidates = randomLightRankerCands.take(selectedCandidateCounter)

selectedLightRankerScribedCandidatesStats.add(selectedCandidates.size.toFloat)

selectedCandidates

}

} else Future.value(candidates)

}

private def lightRankThenTake(

target: Target,

candidates: Seq[CandidateDetails[PushCandidate with TweetCandidate]],

numOfCandidates: Int

): Future[Seq[CandidateDetails[PushCandidate]]] = {

lightRankerCandidateCounter.incr(candidates.length)

lightRankerRequestCounter.incr()

val lightRankerCandidates: Seq[LightRankingCandidate] = candidates.map {

case CandidateDetails(tweetCandidate, \_) =>

val tweetAuthor = tweetCandidate match {

case t: TweetCandidate with TweetAuthor => t.authorId

case \_ => None

}

val hydrationContext: LightRankingFeatureHydrationContext =

LightRankingFeatureHydrationContext.MagicRecsHydrationContext(

MagicRecsFeatureHydrationContext(

tweetAuthor = tweetAuthor,

pushString = tweetCandidate.getPushCopy.flatMap(\_.pushStringGroup).map(\_.toString))

)

LightRankingCandidate(

tweetId = tweetCandidate.tweetId,

hydrationContext = Some(hydrationContext)

)

}

val modelName = target.params(PushFeatureSwitchParams.LightRankingModelTypeParam)

val lightRankedCandidatesFut = {

lightRanker

.rank(UserId(target.targetId), lightRankerCandidates, modelName)

}

lightRankedCandidatesFut.map { lightRankedCandidates =>

val lrScoreMap = lightRankedCandidates.map { lrCand =>

lrCand.tweetId -> lrCand.score

}.toMap

val candScoreMap: Seq[Option[Double]] = candidates.map { candidateDetails =>

lrScoreMap.get(candidateDetails.candidate.tweetId)

}

sortCandidatesByScore(candidates, candScoreMap)

.take(numOfCandidates)

}

}

}