package com.twitter.follow\_recommendations.flows.post\_nux\_ml

import com.google.inject.Inject

import com.google.inject.Singleton

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.IdentityRanker

import com.twitter.follow\_recommendations.common.base.IdentityTransform

import com.twitter.follow\_recommendations.common.base.Ranker

import com.twitter.follow\_recommendations.common.base.Transform

import com.twitter.follow\_recommendations.common.feature\_hydration.common.HasPreFetchedFeature

import com.twitter.follow\_recommendations.common.models.\_

import com.twitter.follow\_recommendations.common.rankers.common.RankerId

import com.twitter.follow\_recommendations.common.rankers.fatigue\_ranker.ImpressionBasedFatigueRanker

import com.twitter.follow\_recommendations.common.rankers.first\_n\_ranker.FirstNRanker

import com.twitter.follow\_recommendations.common.rankers.first\_n\_ranker.FirstNRankerParams

import com.twitter.follow\_recommendations.common.rankers.interleave\_ranker.InterleaveRanker

import com.twitter.follow\_recommendations.common.rankers.ml\_ranker.ranking.HydrateFeaturesTransform

import com.twitter.follow\_recommendations.common.rankers.ml\_ranker.ranking.MlRanker

import com.twitter.follow\_recommendations.common.rankers.ml\_ranker.ranking.MlRankerParams

import com.twitter.follow\_recommendations.common.rankers.weighted\_candidate\_source\_ranker.WeightedCandidateSourceRanker

import com.twitter.follow\_recommendations.configapi.candidates.HydrateCandidateParamsTransform

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.timelines.configapi.HasParams

/\*\*

\* Used to build the combined ranker comprising 4 stages of ranking:

\* - weighted sampler

\* - truncating to the top N merged results for ranking

\* - ML ranker

\* - Interleaving ranker for producer-side experiments

\* - impression-based fatigueing

\*/

@Singleton

class PostNuxMlCombinedRankerBuilder[

T <: HasParams with HasSimilarToContext with HasClientContext with HasExcludedUserIds with HasDisplayLocation with HasDebugOptions with HasPreFetchedFeature with HasDismissedUserIds with HasQualityFactor] @Inject() (

firstNRanker: FirstNRanker[T],

hydrateFeaturesTransform: HydrateFeaturesTransform[T],

hydrateCandidateParamsTransform: HydrateCandidateParamsTransform[T],

mlRanker: MlRanker[T],

statsReceiver: StatsReceiver) {

private[this] val stats: StatsReceiver = statsReceiver.scope("post\_nux\_ml\_ranker")

// we construct each ranker independently and chain them together

def build(

request: T,

candidateSourceWeights: Map[CandidateSourceIdentifier, Double]

): Ranker[T, CandidateUser] = {

val displayLocationStats = stats.scope(request.displayLocation.toString)

val weightedRankerStats: StatsReceiver =

displayLocationStats.scope("weighted\_candidate\_source\_ranker")

val firstNRankerStats: StatsReceiver =

displayLocationStats.scope("first\_n\_ranker")

val hydrateCandidateParamsStats =

displayLocationStats.scope("hydrate\_candidate\_params")

val fatigueRankerStats = displayLocationStats.scope("fatigue\_ranker")

val interleaveRankerStats =

displayLocationStats.scope("interleave\_ranker")

val allRankersStats = displayLocationStats.scope("all\_rankers")

// Checking if the heavy-ranker is an experimental model.

// If it is, InterleaveRanker and candidate parameter hydration are disabled.

// \*NOTE\* that consumer-side experiments should at any time take a small % of traffic, less

// than 20% for instance, to leave enough room for producer experiments. Increasing bucket

// size for producer experiments lead to other issues and is not a viable option for faster

// experiments.

val requestRankerId = request.params(MlRankerParams.RequestScorerIdParam)

if (requestRankerId != RankerId.PostNuxProdRanker) {

hydrateCandidateParamsStats.counter(s"disabled\_by\_${requestRankerId.toString}").incr()

interleaveRankerStats.counter(s"disabled\_by\_${requestRankerId.toString}").incr()

}

// weighted ranker that samples from the candidate sources

val weightedRanker = WeightedCandidateSourceRanker

.build[T](

candidateSourceWeights,

request.params(PostNuxMlParams.CandidateShuffler).shuffle(request.getRandomizationSeed),

randomSeed = request.getRandomizationSeed

).observe(weightedRankerStats)

// ranker that takes the first n results (ie truncates output) while merging duplicates

val firstNRankerObs = firstNRanker.observe(firstNRankerStats)

// either ML ranker that uses deepbirdv2 to score or no ranking

val mainRanker: Ranker[T, CandidateUser] =

buildMainRanker(request, requestRankerId == RankerId.PostNuxProdRanker, displayLocationStats)

// fatigue ranker that uses wtf impressions to fatigue

val fatigueRanker = buildFatigueRanker(request, fatigueRankerStats).observe(fatigueRankerStats)

// interleaveRanker combines rankings from several rankers and enforces candidates' ranks in

// experiment buckets according to their assigned ranker model.

val interleaveRanker =

buildInterleaveRanker(

request,

requestRankerId == RankerId.PostNuxProdRanker,

interleaveRankerStats)

.observe(interleaveRankerStats)

weightedRanker

.andThen(firstNRankerObs)

.andThen(mainRanker)

.andThen(fatigueRanker)

.andThen(interleaveRanker)

.observe(allRankersStats)

}

def buildMainRanker(

request: T,

isMainRankerPostNuxProd: Boolean,

displayLocationStats: StatsReceiver

): Ranker[T, CandidateUser] = {

// note that we may be disabling heavy ranker for users not bucketed

// (due to empty results from the new candidate source)

// need a better solution in the future

val mlRankerStats = displayLocationStats.scope("ml\_ranker")

val noMlRankerStats = displayLocationStats.scope("no\_ml\_ranker")

val hydrateFeaturesStats =

displayLocationStats.scope("hydrate\_features")

val hydrateCandidateParamsStats =

displayLocationStats.scope("hydrate\_candidate\_params")

val notHydrateCandidateParamsStats =

displayLocationStats.scope("not\_hydrate\_candidate\_params")

val rankerStats = displayLocationStats.scope("ranker")

val mlRankerDisabledByExperimentsCounter =

mlRankerStats.counter("disabled\_by\_experiments")

val mlRankerDisabledByQualityFactorCounter =

mlRankerStats.counter("disabled\_by\_quality\_factor")

val disabledByQualityFactor = request.qualityFactor

.exists(\_ <= request.params(PostNuxMlParams.TurnoffMLScorerQFThreshold))

if (disabledByQualityFactor)

mlRankerDisabledByQualityFactorCounter.incr()

if (request.params(PostNuxMlParams.UseMlRanker) && !disabledByQualityFactor) {

val hydrateFeatures = hydrateFeaturesTransform

.observe(hydrateFeaturesStats)

val optionalHydratedParamsTransform: Transform[T, CandidateUser] = {

// We disable candidate parameter hydration for experimental heavy-ranker models.

if (isMainRankerPostNuxProd &&

request.params(PostNuxMlParams.EnableCandidateParamHydration)) {

hydrateCandidateParamsTransform

.observe(hydrateCandidateParamsStats)

} else {

new IdentityTransform[T, CandidateUser]()

.observe(notHydrateCandidateParamsStats)

}

}

val candidateSize = request.params(FirstNRankerParams.CandidatesToRank)

Ranker

.chain(

hydrateFeatures.andThen(optionalHydratedParamsTransform),

mlRanker.observe(mlRankerStats),

)

.within(

request.params(PostNuxMlParams.MlRankerBudget),

rankerStats.scope(s"n$candidateSize"))

} else {

new IdentityRanker[T, CandidateUser].observe(noMlRankerStats)

}

}

def buildInterleaveRanker(

request: T,

isMainRankerPostNuxProd: Boolean,

interleaveRankerStats: StatsReceiver

): Ranker[T, CandidateUser] = {

// InterleaveRanker is enabled only for display locations powered by the PostNux heavy-ranker.

if (request.params(PostNuxMlParams.EnableInterleaveRanker) &&

// InterleaveRanker is disabled for requests with experimental heavy-rankers.

isMainRankerPostNuxProd) {

new InterleaveRanker[T](interleaveRankerStats)

} else {

new IdentityRanker[T, CandidateUser]()

}

}

def buildFatigueRanker(

request: T,

fatigueRankerStats: StatsReceiver

): Ranker[T, CandidateUser] = {

if (request.params(PostNuxMlParams.EnableFatigueRanker)) {

ImpressionBasedFatigueRanker

.build[T](

fatigueRankerStats

).within(request.params(PostNuxMlParams.FatigueRankerBudget), fatigueRankerStats)

} else {

new IdentityRanker[T, CandidateUser]()

}

}

}