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

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.EnrichedCandidateSource.\_

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

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

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

import com.twitter.follow\_recommendations.common.predicates.dismiss.DismissedCandidatePredicate

import com.twitter.follow\_recommendations.common.predicates.gizmoduck.GizmoduckPredicate

import com.twitter.follow\_recommendations.common.transforms.ranker\_id.RandomRankerIdTransform

import com.twitter.follow\_recommendations.common.predicates.sgs.InvalidTargetCandidateRelationshipTypesPredicate

import com.twitter.follow\_recommendations.common.predicates.sgs.RecentFollowingPredicate

import com.twitter.follow\_recommendations.common.predicates.CandidateParamPredicate

import com.twitter.follow\_recommendations.common.predicates.CandidateSourceParamPredicate

import com.twitter.follow\_recommendations.common.predicates.CuratedCompetitorListPredicate

import com.twitter.follow\_recommendations.common.predicates.ExcludedUserIdPredicate

import com.twitter.follow\_recommendations.common.predicates.InactivePredicate

import com.twitter.follow\_recommendations.common.predicates.PreviouslyRecommendedUserIdsPredicate

import com.twitter.follow\_recommendations.common.predicates.user\_activity.NonNearZeroUserActivityPredicate

import com.twitter.follow\_recommendations.common.transforms.dedup.DedupTransform

import com.twitter.follow\_recommendations.common.transforms.modify\_social\_proof.ModifySocialProofTransform

import com.twitter.follow\_recommendations.common.transforms.tracking\_token.TrackingTokenTransform

import com.twitter.follow\_recommendations.common.transforms.weighted\_sampling.SamplingTransform

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

import com.twitter.follow\_recommendations.configapi.params.GlobalParams

import com.twitter.follow\_recommendations.configapi.params.GlobalParams.EnableGFSSocialProofTransform

import com.twitter.follow\_recommendations.utils.CandidateSourceHoldbackUtil

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

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

import com.twitter.timelines.configapi.Params

import com.twitter.util.Duration

import javax.inject.Inject

import javax.inject.Singleton

import com.twitter.follow\_recommendations.common.clients.socialgraph.SocialGraphClient

import com.twitter.follow\_recommendations.common.predicates.hss.HssPredicate

import com.twitter.follow\_recommendations.common.predicates.sgs.InvalidRelationshipPredicate

import com.twitter.follow\_recommendations.common.transforms.modify\_social\_proof.RemoveAccountProofTransform

import com.twitter.follow\_recommendations.logging.FrsLogger

import com.twitter.follow\_recommendations.models.RecommendationFlowData

import com.twitter.follow\_recommendations.utils.RecommendationFlowBaseSideEffectsUtil

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

import com.twitter.product\_mixer.core.quality\_factor.BoundsWithDefault

import com.twitter.product\_mixer.core.quality\_factor.LinearLatencyQualityFactor

import com.twitter.product\_mixer.core.quality\_factor.LinearLatencyQualityFactorConfig

import com.twitter.product\_mixer.core.quality\_factor.LinearLatencyQualityFactorObserver

import com.twitter.product\_mixer.core.quality\_factor.QualityFactorObserver

import com.twitter.stitch.Stitch

/\*\*

\* We use this flow for all post-nux display locations that would use a machine-learning-based-ranker

\* eg HTL, Sidebar, etc

\* Note that the RankedPostNuxFlow is used primarily for scribing/data collection, and doesn't

\* incorporate all of the other components in a flow (candidate source generation, predicates etc)

\*/

@Singleton

class PostNuxMlFlow @Inject() (

postNuxMlCandidateSourceRegistry: PostNuxMlCandidateSourceRegistry,

postNuxMlCombinedRankerBuilder: PostNuxMlCombinedRankerBuilder[PostNuxMlRequest],

curatedCompetitorListPredicate: CuratedCompetitorListPredicate,

gizmoduckPredicate: GizmoduckPredicate,

sgsPredicate: InvalidTargetCandidateRelationshipTypesPredicate,

hssPredicate: HssPredicate,

invalidRelationshipPredicate: InvalidRelationshipPredicate,

recentFollowingPredicate: RecentFollowingPredicate,

nonNearZeroUserActivityPredicate: NonNearZeroUserActivityPredicate,

inactivePredicate: InactivePredicate,

dismissedCandidatePredicate: DismissedCandidatePredicate,

previouslyRecommendedUserIdsPredicate: PreviouslyRecommendedUserIdsPredicate,

modifySocialProofTransform: ModifySocialProofTransform,

removeAccountProofTransform: RemoveAccountProofTransform,

trackingTokenTransform: TrackingTokenTransform,

randomRankerIdTransform: RandomRankerIdTransform,

candidateParamsFactory: CandidateUserParamsFactory[PostNuxMlRequest],

samplingTransform: SamplingTransform,

frsLogger: FrsLogger,

baseStatsReceiver: StatsReceiver)

extends RecommendationFlow[PostNuxMlRequest, CandidateUser]

with RecommendationFlowBaseSideEffectsUtil[PostNuxMlRequest, CandidateUser]

with CandidateSourceHoldbackUtil {

override protected val targetEligibility: Predicate[PostNuxMlRequest] =

new ParamPredicate[PostNuxMlRequest](PostNuxMlParams.TargetEligibility)

override val statsReceiver: StatsReceiver = baseStatsReceiver.scope("post\_nux\_ml\_flow")

override val qualityFactorObserver: Option[QualityFactorObserver] = {

val config = LinearLatencyQualityFactorConfig(

qualityFactorBounds =

BoundsWithDefault(minInclusive = 0.1, maxInclusive = 1.0, default = 1.0),

initialDelay = 60.seconds,

targetLatency = 700.milliseconds,

targetLatencyPercentile = 95.0,

delta = 0.001

)

val qualityFactor = LinearLatencyQualityFactor(config)

val observer = LinearLatencyQualityFactorObserver(qualityFactor)

statsReceiver.provideGauge("quality\_factor")(qualityFactor.currentValue.toFloat)

Some(observer)

}

override protected def updateTarget(request: PostNuxMlRequest): Stitch[PostNuxMlRequest] = {

Stitch.value(

request.copy(qualityFactor = qualityFactorObserver.map(\_.qualityFactor.currentValue))

)

}

private[post\_nux\_ml] def getCandidateSourceIdentifiers(

params: Params

): Set[CandidateSourceIdentifier] = {

PostNuxMlFlowCandidateSourceWeights.getWeights(params).keySet

}

override protected def candidateSources(

request: PostNuxMlRequest

): Seq[CandidateSource[PostNuxMlRequest, CandidateUser]] = {

val identifiers = getCandidateSourceIdentifiers(request.params)

val selected: Set[CandidateSource[PostNuxMlRequest, CandidateUser]] =

postNuxMlCandidateSourceRegistry.select(identifiers)

val budget: Duration = request.params(PostNuxMlParams.FetchCandidateSourceBudget)

filterCandidateSources(

request,

selected.map(c => c.failOpenWithin(budget, statsReceiver)).toSeq)

}

override protected val preRankerCandidateFilter: Predicate[(PostNuxMlRequest, CandidateUser)] = {

val stats = statsReceiver.scope("pre\_ranker")

object excludeNearZeroUserPredicate

extends GatedPredicateBase[(PostNuxMlRequest, CandidateUser)](

nonNearZeroUserActivityPredicate,

stats.scope("exclude\_near\_zero\_predicate")

) {

override def gate(item: (PostNuxMlRequest, CandidateUser)): Boolean =

item.\_1.params(PostNuxMlParams.ExcludeNearZeroCandidates)

}

object invalidRelationshipGatedPredicate

extends GatedPredicateBase[(PostNuxMlRequest, CandidateUser)](

invalidRelationshipPredicate,

stats.scope("invalid\_relationship\_predicate")

) {

override def gate(item: (PostNuxMlRequest, CandidateUser)): Boolean =

item.\_1.params(PostNuxMlParams.EnableInvalidRelationshipPredicate)

}

ExcludedUserIdPredicate

.observe(stats.scope("exclude\_user\_id\_predicate"))

.andThen(

recentFollowingPredicate.observe(stats.scope("recent\_following\_predicate"))

)

.andThen(

dismissedCandidatePredicate.observe(stats.scope("dismissed\_candidate\_predicate"))

)

.andThen(

previouslyRecommendedUserIdsPredicate.observe(

stats.scope("previously\_recommended\_user\_ids\_predicate"))

)

.andThen(

invalidRelationshipGatedPredicate.observe(stats.scope("invalid\_relationship\_predicate"))

)

.andThen(

excludeNearZeroUserPredicate.observe(stats.scope("exclude\_near\_zero\_user\_state"))

)

.observe(stats.scope("overall\_pre\_ranker\_candidate\_filter"))

}

override protected def selectRanker(

request: PostNuxMlRequest

): Ranker[PostNuxMlRequest, CandidateUser] = {

postNuxMlCombinedRankerBuilder.build(

request,

PostNuxMlFlowCandidateSourceWeights.getWeights(request.params))

}

override protected val postRankerTransform: Transform[PostNuxMlRequest, CandidateUser] = {

new DedupTransform[PostNuxMlRequest, CandidateUser]

.observe(statsReceiver.scope("dedupping"))

.andThen(

samplingTransform

.gated(PostNuxMlParams.SamplingTransformEnabled)

.observe(statsReceiver.scope("samplingtransform")))

}

override protected val validateCandidates: Predicate[(PostNuxMlRequest, CandidateUser)] = {

val stats = statsReceiver.scope("validate\_candidates")

val competitorPredicate =

curatedCompetitorListPredicate.map[(PostNuxMlRequest, CandidateUser)](\_.\_2)

val producerHoldbackPredicate = new CandidateParamPredicate[CandidateUser](

GlobalParams.KeepUserCandidate,

FilterReason.CandidateSideHoldback

).map[(PostNuxMlRequest, CandidateUser)] {

case (request, user) => candidateParamsFactory(user, request)

}

val pymkProducerHoldbackPredicate = new CandidateSourceParamPredicate(

GlobalParams.KeepSocialUserCandidate,

FilterReason.CandidateSideHoldback,

CandidateSourceHoldbackUtil.SocialCandidateSourceIds

).map[(PostNuxMlRequest, CandidateUser)] {

case (request, user) => candidateParamsFactory(user, request)

}

val sgsPredicateStats = stats.scope("sgs\_predicate")

object sgsGatedPredicate

extends GatedPredicateBase[(PostNuxMlRequest, CandidateUser)](

sgsPredicate.observe(sgsPredicateStats),

sgsPredicateStats

) {

/\*\*

\* When SGS predicate is turned off, only query SGS exists API for (user, candidate, relationship)

\* when the user's number of invalid relationships exceeds the threshold during request

\* building step. This is to minimize load to SGS and underlying Flock DB.

\*/

override def gate(item: (PostNuxMlRequest, CandidateUser)): Boolean =

item.\_1.params(PostNuxMlParams.EnableSGSPredicate) ||

SocialGraphClient.enablePostRankerSgsPredicate(

item.\_1.invalidRelationshipUserIds.getOrElse(Set.empty).size)

}

val hssPredicateStats = stats.scope("hss\_predicate")

object hssGatedPredicate

extends GatedPredicateBase[(PostNuxMlRequest, CandidateUser)](

hssPredicate.observe(hssPredicateStats),

hssPredicateStats

) {

override def gate(item: (PostNuxMlRequest, CandidateUser)): Boolean =

item.\_1.params(PostNuxMlParams.EnableHssPredicate)

}

Predicate

.andConcurrently[(PostNuxMlRequest, CandidateUser)](

Seq(

competitorPredicate.observe(stats.scope("curated\_competitor\_predicate")),

gizmoduckPredicate.observe(stats.scope("gizmoduck\_predicate")),

sgsGatedPredicate,

hssGatedPredicate,

inactivePredicate.observe(stats.scope("inactive\_predicate")),

)

)

// to avoid dilutions, we need to apply the receiver holdback predicates at the very last step

.andThen(pymkProducerHoldbackPredicate.observe(stats.scope("pymk\_receiver\_side\_holdback")))

.andThen(producerHoldbackPredicate.observe(stats.scope("receiver\_side\_holdback")))

.observe(stats.scope("overall\_validate\_candidates"))

}

override protected val transformResults: Transform[PostNuxMlRequest, CandidateUser] = {

modifySocialProofTransform

.gated(EnableGFSSocialProofTransform)

.andThen(trackingTokenTransform)

.andThen(randomRankerIdTransform.gated(PostNuxMlParams.LogRandomRankerId))

.andThen(removeAccountProofTransform.gated(PostNuxMlParams.EnableRemoveAccountProofTransform))

}

override protected def resultsConfig(request: PostNuxMlRequest): RecommendationResultsConfig = {

RecommendationResultsConfig(

request.maxResults.getOrElse(request.params(PostNuxMlParams.ResultSizeParam)),

request.params(PostNuxMlParams.BatchSizeParam)

)

}

override def applySideEffects(

target: PostNuxMlRequest,

candidateSources: Seq[CandidateSource[PostNuxMlRequest, CandidateUser]],

candidatesFromCandidateSources: Seq[CandidateUser],

mergedCandidates: Seq[CandidateUser],

filteredCandidates: Seq[CandidateUser],

rankedCandidates: Seq[CandidateUser],

transformedCandidates: Seq[CandidateUser],

truncatedCandidates: Seq[CandidateUser],

results: Seq[CandidateUser]

): Stitch[Unit] = {

frsLogger.logRecommendationFlowData[PostNuxMlRequest](

target,

RecommendationFlowData[PostNuxMlRequest](

target,

PostNuxMlFlow.identifier,

candidateSources,

candidatesFromCandidateSources,

mergedCandidates,

filteredCandidates,

rankedCandidates,

transformedCandidates,

truncatedCandidates,

results

)

)

super.applySideEffects(

target,

candidateSources,

candidatesFromCandidateSources,

mergedCandidates,

filteredCandidates,

rankedCandidates,

transformedCandidates,

truncatedCandidates,

results

)

}

}

object PostNuxMlFlow {

val identifier = RecommendationPipelineIdentifier("PostNuxMlFlow")

}