package com.twitter.frigate.pushservice.ml

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.common.ml.feature.TweetSocialProofKey

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

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

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

import com.twitter.frigate.pushservice.predicate.quality\_model\_predicate.PDauCohortUtil

import com.twitter.nrel.hydration.base.FeatureInput

import com.twitter.nrel.hydration.push.HydrationContext

import com.twitter.nrel.hydration.frigate.{FeatureInputs => FI}

import com.twitter.util.Future

object HydrationContextBuilder {

private def getRecUserInputs(

pushCandidate: PushCandidate

): Set[FI.RecUser] = {

pushCandidate match {

case userCandidate: UserCandidate =>

Set(FI.RecUser(userCandidate.userId))

case \_ => Set.empty

}

}

private def getRecTweetInputs(

pushCandidate: PushCandidate

): Set[FI.RecTweet] =

pushCandidate match {

case tweetCandidateWithAuthor: TweetCandidate with TweetAuthor with TweetAuthorDetails =>

val authorIdOpt = tweetCandidateWithAuthor.authorId

Set(FI.RecTweet(tweetCandidateWithAuthor.tweetId, authorIdOpt))

case \_ => Set.empty

}

private def getMediaInputs(

pushCandidate: PushCandidate

): Set[FI.Media] =

pushCandidate match {

case tweetCandidateWithMedia: TweetCandidate with TweetDetails =>

tweetCandidateWithMedia.mediaKeys

.map { mk =>

Set(FI.Media(mk))

}.getOrElse(Set.empty)

case \_ => Set.empty

}

private def getEventInputs(

pushCandidate: PushCandidate

): Set[FI.Event] = pushCandidate match {

case mrEventCandidate: EventCandidate =>

Set(FI.Event(mrEventCandidate.eventId))

case mfEventCandidate: MagicFanoutEventCandidate =>

Set(FI.Event(mfEventCandidate.eventId))

case \_ => Set.empty

}

private def getTopicInputs(

pushCandidate: PushCandidate

): Set[FI.Topic] =

pushCandidate match {

case mrTopicCandidate: TopicCandidate =>

mrTopicCandidate.semanticCoreEntityId match {

case Some(topicId) => Set(FI.Topic(topicId))

case \_ => Set.empty

}

case \_ => Set.empty

}

private def getTweetSocialProofKey(

pushCandidate: PushCandidate

): Future[Set[FI.SocialProofKey]] = {

pushCandidate match {

case candidate: TweetCandidate with SocialContextActions =>

val target = pushCandidate.target

target.seedsWithWeight.map { seedsWithWeightOpt =>

Set(

FI.SocialProofKey(

TweetSocialProofKey(

seedsWithWeightOpt.getOrElse(Map.empty),

candidate.socialContextAllTypeActions

))

)

}

case \_ => Future.value(Set.empty)

}

}

private def getSocialContextInputs(

pushCandidate: PushCandidate

): Future[Set[FeatureInput]] =

pushCandidate match {

case candidateWithSC: Candidate with SocialContextActions =>

val tweetSocialProofKeyFut = getTweetSocialProofKey(pushCandidate)

tweetSocialProofKeyFut.map { tweetSocialProofKeyOpt =>

val socialContextUsers = FI.SocialContextUsers(candidateWithSC.socialContextUserIds.toSet)

val socialContextActions =

FI.SocialContextActions(candidateWithSC.socialContextAllTypeActions)

val socialProofKeyOpt = tweetSocialProofKeyOpt

Set(Set(socialContextUsers), Set(socialContextActions), socialProofKeyOpt).flatten

}

case \_ => Future.value(Set.empty)

}

private def getPushStringGroupInputs(

pushCandidate: PushCandidate

): Set[FI.PushStringGroup] =

Set(

FI.PushStringGroup(

pushCandidate.getPushCopy.flatMap(\_.pushStringGroup).map(\_.toString).getOrElse("")

))

private def getCRTInputs(

pushCandidate: PushCandidate

): Set[FI.CommonRecommendationType] =

Set(FI.CommonRecommendationType(pushCandidate.commonRecType))

private def getFrigateNotification(

pushCandidate: PushCandidate

): Set[FI.CandidateFrigateNotification] =

Set(FI.CandidateFrigateNotification(pushCandidate.frigateNotification))

private def getCopyId(

pushCandidate: PushCandidate

): Set[FI.CopyId] =

Set(FI.CopyId(pushCandidate.pushCopyId, pushCandidate.ntabCopyId))

def build(candidate: PushCandidate): Future[HydrationContext] = {

val socialContextInputsFut = getSocialContextInputs(candidate)

socialContextInputsFut.map { socialContextInputs =>

val featureInputs: Set[FeatureInput] =

socialContextInputs ++

getRecUserInputs(candidate) ++

getRecTweetInputs(candidate) ++

getEventInputs(candidate) ++

getTopicInputs(candidate) ++

getCRTInputs(candidate) ++

getPushStringGroupInputs(candidate) ++

getMediaInputs(candidate) ++

getFrigateNotification(candidate) ++

getCopyId(candidate)

HydrationContext(

candidate.target.targetId,

featureInputs

)

}

}

def build(target: Target): Future[HydrationContext] = {

val realGraphFeaturesFut = target.realGraphFeatures

for {

realGraphFeaturesOpt <- realGraphFeaturesFut

dauProb <- PDauCohortUtil.getDauProb(target)

mrUserStateOpt <- target.targetMrUserState

historyInputOpt <-

if (target.params(PushFeatureSwitchParams.EnableHydratingOnlineMRHistoryFeatures)) {

target.onlineLabeledPushRecs.map { mrHistoryValueOpt =>

mrHistoryValueOpt.map(FI.MrHistory)

}

} else Future.None

} yield {

val realGraphFeaturesInputOpt = realGraphFeaturesOpt.map { realGraphFeatures =>

FI.TargetRealGraphFeatures(realGraphFeatures)

}

val dauProbInput = FI.DauProb(dauProb)

val mrUserStateInput = FI.MrUserState(mrUserStateOpt.map(\_.name).getOrElse("unknown"))

HydrationContext(

target.targetId,

Seq(

realGraphFeaturesInputOpt,

historyInputOpt,

Some(dauProbInput),

Some(mrUserStateInput)

).flatten.toSet

)

}

}

}