package com.twitter.frigate.pushservice.target

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.frigate.common.base.TargetUser

import com.twitter.frigate.common.candidate.TargetABDecider

import com.twitter.frigate.common.candidate.TargetDecider

import com.twitter.frigate.common.candidate.UserDetails

import com.twitter.frigate.data\_pipeline.thriftscala.UserHistoryValue

import com.twitter.frigate.dau\_model.thriftscala.DauProbability

import com.twitter.frigate.scribe.thriftscala.SkipModelInfo

import com.twitter.hermit.stp.thriftscala.STPResult

import com.twitter.timelines.real\_graph.v1.thriftscala.RealGraphFeatures

import com.twitter.util.Future

import com.twitter.util.Time

import com.twitter.frigate.pushservice.params.DeciderKey

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

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

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

import com.twitter.frigate.pushservice.util.PushDeviceUtil

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

import com.twitter.timelines.configapi.FSParam

trait TargetScoringDetails {

tuc: TargetUser with TargetDecider with TargetABDecider with UserDetails =>

def stats: StatsReceiver

/\*

\* We have 3 types of model training data:

\* 1, skip ranker and model predicates

\* controlled by decider frigate\_notifier\_quality\_model\_training\_data

\* the data distribution is same to the distribution in ranking

\* 2, skip model predicates only

\* controlled by decider skip\_ml\_model\_predicate

\* the data distribution is same to the distribution in filtering

\* 3, no skip, only scribe features

\* controlled by decider scribe\_model\_features

\* the data distribution is same to production traffic

\* The "miscellaneous" is used to store all misc information for selecting the data offline (e.g., ddg-bucket information)

\* \*/

lazy val skipModelInfo: Option[SkipModelInfo] = {

val trainingDataDeciderKey = DeciderKey.trainingDataDeciderKey.toString

val skipMlModelPredicateDeciderKey = DeciderKey.skipMlModelPredicateDeciderKey.toString

val scribeModelFeaturesDeciderKey = DeciderKey.scribeModelFeaturesDeciderKey.toString

val miscellaneous = None

if (isDeciderEnabled(trainingDataDeciderKey, stats, useRandomRecipient = true)) {

Some(

SkipModelInfo(

skipPushOpenPredicate = Some(true),

skipPushRanker = Some(true),

miscellaneous = miscellaneous))

} else if (isDeciderEnabled(skipMlModelPredicateDeciderKey, stats, useRandomRecipient = true)) {

Some(

SkipModelInfo(

skipPushOpenPredicate = Some(true),

skipPushRanker = Some(false),

miscellaneous = miscellaneous))

} else if (isDeciderEnabled(scribeModelFeaturesDeciderKey, stats, useRandomRecipient = true)) {

Some(SkipModelInfo(noSkipButScribeFeatures = Some(true), miscellaneous = miscellaneous))

} else {

Some(SkipModelInfo(miscellaneous = miscellaneous))

}

}

lazy val scribeFeatureForRequestScribe =

isDeciderEnabled(

DeciderKey.scribeModelFeaturesForRequestScribe.toString,

stats,

useRandomRecipient = true)

lazy val rankingModelParam: Future[FSParam[WeightedOpenOrNtabClickModel.ModelNameType]] =

tuc.deviceInfo.map { deviceInfoOpt =>

if (PushDeviceUtil.isPrimaryDeviceAndroid(deviceInfoOpt) &&

tuc.params(PushParams.AndroidOnlyRankingExperimentParam)) {

PushFeatureSwitchParams.WeightedOpenOrNtabClickRankingModelForAndroidParam

} else {

PushFeatureSwitchParams.WeightedOpenOrNtabClickRankingModelParam

}

}

lazy val filteringModelParam: FSParam[WeightedOpenOrNtabClickModel.ModelNameType] =

PushFeatureSwitchParams.WeightedOpenOrNtabClickFilteringModelParam

def skipMlRanker: Boolean = skipModelInfo.exists(\_.skipPushRanker.contains(true))

def skipModelPredicate: Boolean = skipModelInfo.exists(\_.skipPushOpenPredicate.contains(true))

def noSkipButScribeFeatures: Boolean =

skipModelInfo.exists(\_.noSkipButScribeFeatures.contains(true))

def isModelTrainingData: Boolean = skipMlRanker || skipModelPredicate || noSkipButScribeFeatures

def scribeFeatureWithoutHydratingNewFeatures: Boolean =

isDeciderEnabled(

DeciderKey.scribeModelFeaturesWithoutHydratingNewFeaturesDeciderKey.toString,

stats,

useRandomRecipient = true

)

def targetHydrationContext: Future[HydrationContext]

def featureMap: Future[FeatureMap]

def dauProbability: Future[Option[DauProbability]]

def labeledPushRecsHydrated: Future[Option[UserHistoryValue]]

def onlineLabeledPushRecs: Future[Option[UserHistoryValue]]

def realGraphFeatures: Future[Option[RealGraphFeatures]]

def stpResult: Future[Option[STPResult]]

def globalOptoutProbabilities: Seq[Future[Option[Double]]]

def bucketOptoutProbability: Future[Option[Double]]

val sendTime: Long = Time.now.inMillis

}