package com.twitter.follow\_recommendations.common.candidate\_sources.ppmi\_locale\_follow

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.candidate\_sources.ppmi\_locale\_follow.PPMILocaleFollowSourceParams.CandidateSourceEnabled

import com.twitter.follow\_recommendations.common.candidate\_sources.ppmi\_locale\_follow.PPMILocaleFollowSourceParams.LocaleToExcludeFromRecommendation

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

import com.twitter.hermit.model.Algorithm

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

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

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

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

import com.twitter.strato.generated.client.onboarding.UserPreferredLanguagesOnUserClientColumn

import com.twitter.strato.generated.client.onboarding.userrecs.LocaleFollowPpmiClientColumn

import com.twitter.timelines.configapi.HasParams

/\*\*

\* Fetches candidates based on the Positive Pointwise Mutual Information (PPMI) statistic

\* for a set of locales

\* \*/

@Singleton

class PPMILocaleFollowSource @Inject() (

userPreferredLanguagesOnUserClientColumn: UserPreferredLanguagesOnUserClientColumn,

localeFollowPpmiClientColumn: LocaleFollowPpmiClientColumn,

statsReceiver: StatsReceiver)

extends CandidateSource[HasClientContext with HasParams, CandidateUser] {

override val identifier: CandidateSourceIdentifier = PPMILocaleFollowSource.Identifier

private val stats = statsReceiver.scope("PPMILocaleFollowSource")

override def apply(target: HasClientContext with HasParams): Stitch[Seq[CandidateUser]] = {

(for {

countryCode <- target.getCountryCode

userId <- target.getOptionalUserId

} yield {

getPreferredLocales(userId, countryCode.toLowerCase())

.flatMap { locale =>

stats.addGauge("allLocale") {

locale.length

}

val filteredLocale =

locale.filter(!target.params(LocaleToExcludeFromRecommendation).contains(\_))

stats.addGauge("postFilterLocale") {

filteredLocale.length

}

if (target.params(CandidateSourceEnabled)) {

getPPMILocaleFollowCandidates(filteredLocale)

} else Stitch(Seq.empty)

}

.map(\_.sortBy(\_.score)(Ordering[Option[Double]].reverse)

.take(PPMILocaleFollowSource.DefaultMaxCandidatesToReturn))

}).getOrElse(Stitch.Nil)

}

private def getPPMILocaleFollowCandidates(

locales: Seq[String]

): Stitch[Seq[CandidateUser]] = {

Stitch

.traverse(locales) { locale =>

// Get PPMI candidates for each locale

localeFollowPpmiClientColumn.fetcher

.fetch(locale)

.map(\_.v

.map(\_.candidates).getOrElse(Nil).map { candidate =>

CandidateUser(id = candidate.userId, score = Some(candidate.score))

}.map(\_.withCandidateSource(identifier)))

}.map(\_.flatten)

}

private def getPreferredLocales(userId: Long, countryCode: String): Stitch[Seq[String]] = {

userPreferredLanguagesOnUserClientColumn.fetcher

.fetch(userId)

.map(\_.v.map(\_.languages).getOrElse(Nil).map { lang =>

s"$countryCode-$lang".toLowerCase

})

}

}

object PPMILocaleFollowSource {

val Identifier = CandidateSourceIdentifier(Algorithm.PPMILocaleFollow.toString)

val DefaultMaxCandidatesToReturn = 100

}