package com.twitter.cr\_mixer.filter

import com.twitter.cr\_mixer.model.CrCandidateGeneratorQuery

import com.twitter.cr\_mixer.model.RankedCandidate

import com.twitter.cr\_mixer.thriftscala.SourceType

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class PostRankFilterRunner @Inject() (

globalStats: StatsReceiver) {

private val scopedStats = globalStats.scope(this.getClass.getCanonicalName)

private val beforeCount = scopedStats.stat("candidate\_count", "before")

private val afterCount = scopedStats.stat("candidate\_count", "after")

def run(

query: CrCandidateGeneratorQuery,

candidates: Seq[RankedCandidate]

): Future[Seq[RankedCandidate]] = {

beforeCount.add(candidates.size)

Future(

removeBadRecentNotificationCandidates(candidates)

).map { results =>

afterCount.add(results.size)

results

}

}

/\*\*

\* Remove "bad" quality candidates generated by recent notifications

\* A candidate is bad when it is generated by a single RecentNotification

\* SourceKey.

\* e.x:

\* tweetA {recent notification1} -> bad

\* tweetB {recent notification1 recent notification2} -> good

\*tweetC {recent notification1 recent follow1} -> bad

\* SD-19397

\*/

private[filter] def removeBadRecentNotificationCandidates(

candidates: Seq[RankedCandidate]

): Seq[RankedCandidate] = {

candidates.filterNot {

isBadQualityRecentNotificationCandidate

}

}

private def isBadQualityRecentNotificationCandidate(candidate: RankedCandidate): Boolean = {

candidate.potentialReasons.size == 1 &&

candidate.potentialReasons.head.sourceInfoOpt.nonEmpty &&

candidate.potentialReasons.head.sourceInfoOpt.get.sourceType == SourceType.NotificationClick

}

}