package com.twitter.timelineranker.common

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FutureArrow

import com.twitter.timelineranker.core.CandidateEnvelope

import com.twitter.timelineranker.model.RecapQuery.DependencyProvider

import com.twitter.timelineranker.util.SearchResultUtil

import com.twitter.util.Future

/\*\*

\* Truncate the search results by score. Assumes that the search results are sorted in

\* score-descending order unless extraSortBeforeTruncation is set to true.

\*

\* This transform has two main use cases:

\*

\* - when returnAllResults is set to true, earlybird returns (numResultsPerShard \* number of shards)

\* results. this transform is then used to further truncate the result, so that the size will be the

\* same as when returnAllResults is set to false.

\*

\* - we retrieve extra number of results from earlybird, as specified in MaxCountMultiplierParam,

\* so that we are left with sufficient number of candidates after hydration and filtering.

\* this transform will be used to get rid of extra results we ended up not using.

\*/

class RecapSearchResultsTruncationTransform(

extraSortBeforeTruncationGate: DependencyProvider[Boolean],

maxCountProvider: DependencyProvider[Int],

statsReceiver: StatsReceiver)

extends FutureArrow[CandidateEnvelope, CandidateEnvelope] {

private[this] val postTruncationSizeStat = statsReceiver.stat("postTruncationSize")

private[this] val earlybirdScoreX100Stat = statsReceiver.stat("earlybirdScoreX100")

override def apply(envelope: CandidateEnvelope): Future[CandidateEnvelope] = {

val sortBeforeTruncation = extraSortBeforeTruncationGate(envelope.query)

val maxCount = maxCountProvider(envelope.query)

val searchResults = envelope.searchResults

// set aside results that are marked by isRandomTweet field

val (randomTweetSeq, searchResultsExcludingRandom) = searchResults.partition { result =>

result.tweetFeatures.flatMap(\_.isRandomTweet).getOrElse(false)

}

// sort and truncate searchResults other than the random tweet

val maxCountExcludingRandom = Math.max(0, maxCount - randomTweetSeq.size)

val truncatedResultsExcludingRandom =

if (sortBeforeTruncation || searchResultsExcludingRandom.size > maxCountExcludingRandom) {

val sorted = if (sortBeforeTruncation) {

searchResultsExcludingRandom.sortWith(

SearchResultUtil.getScore(\_) > SearchResultUtil.getScore(\_))

} else searchResultsExcludingRandom

sorted.take(maxCountExcludingRandom)

} else searchResultsExcludingRandom

// put back the random tweet set aside previously

val allTruncatedResults = truncatedResultsExcludingRandom ++ randomTweetSeq

// stats

postTruncationSizeStat.add(allTruncatedResults.size)

allTruncatedResults.foreach { result =>

val earlybirdScoreX100 =

result.metadata.flatMap(\_.score).getOrElse(0.0).toFloat \* 100

earlybirdScoreX100Stat.add(earlybirdScoreX100)

}

Future.value(envelope.copy(searchResults = allTruncatedResults))

}

}