package com.twitter.timelineranker.uteg\_liked\_by\_tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.recos.user\_tweet\_entity\_graph.thriftscala.TweetRecommendation

import com.twitter.search.earlybird.thriftscala.ThriftSearchResult

import com.twitter.search.earlybird.thriftscala.ThriftSearchResultMetadata

import com.twitter.servo.util.FutureArrow

import com.twitter.timelineranker.core.CandidateEnvelope

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

import com.twitter.timelines.model.TweetId

import com.twitter.util.Future

object CombinedScoreAndTruncateTransform {

val DefaultRealGraphWeight = 1.0

val DefaultEmptyScore = 0.0

}

/\*\*

\* Rank and truncate search results according to

\* DefaultRealGraphWeight \* real\_graph\_score + earlybird\_score\_multiplier \* earlybird\_score

\* Note: scoring and truncation only applies to out of network candidates

\*/

class CombinedScoreAndTruncateTransform(

maxTweetCountProvider: DependencyProvider[Int],

earlybirdScoreMultiplierProvider: DependencyProvider[Double],

numAdditionalRepliesProvider: DependencyProvider[Int],

statsReceiver: StatsReceiver)

extends FutureArrow[CandidateEnvelope, CandidateEnvelope] {

import CombinedScoreAndTruncateTransform.\_

private[this] val scopedStatsReceiver = statsReceiver.scope("CombinedScoreAndTruncateTransform")

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

private[this] val realGraphScoreX100Stat = scopedStatsReceiver.stat("realGraphScoreX100")

private[this] val additionalReplyCounter = scopedStatsReceiver.counter("additionalReplies")

private[this] val resultCounter = scopedStatsReceiver.counter("results")

private[this] def getRealGraphScore(

searchResult: ThriftSearchResult,

utegResults: Map[TweetId, TweetRecommendation]

): Double = {

utegResults.get(searchResult.id).map(\_.score).getOrElse(DefaultEmptyScore)

}

private[this] def getEarlybirdScore(metadataOpt: Option[ThriftSearchResultMetadata]): Double = {

metadataOpt

.flatMap(metadata => metadata.score)

.getOrElse(DefaultEmptyScore)

}

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

val maxCount = maxTweetCountProvider(envelope.query)

val earlybirdScoreMultiplier = earlybirdScoreMultiplierProvider(envelope.query)

val realGraphScoreMultiplier = DefaultRealGraphWeight

val searchResultsAndScore = envelope.searchResults.map { searchResult =>

val realGraphScore = getRealGraphScore(searchResult, envelope.utegResults)

val earlybirdScore = getEarlybirdScore(searchResult.metadata)

earlybirdScoreX100Stat.add(earlybirdScore.toFloat \* 100)

realGraphScoreX100Stat.add(realGraphScore.toFloat \* 100)

val combinedScore =

realGraphScoreMultiplier \* realGraphScore + earlybirdScoreMultiplier \* earlybirdScore

(searchResult, combinedScore)

}

// set aside results that are marked by isRandomTweet field

val (randomSearchResults, otherSearchResults) = searchResultsAndScore.partition {

resultAndScore =>

resultAndScore.\_1.tweetFeatures.flatMap(\_.isRandomTweet).getOrElse(false)

}

val (topResults, remainingResults) = otherSearchResults

.sortBy(\_.\_2)(Ordering[Double].reverse).map(\_.\_1).splitAt(

maxCount - randomSearchResults.length)

val numAdditionalReplies = numAdditionalRepliesProvider(envelope.query)

val additionalReplies = {

if (numAdditionalReplies > 0) {

val replyTweetIdSet =

envelope.hydratedTweets.outerTweets.filter(\_.hasReply).map(\_.tweetId).toSet

remainingResults.filter(result => replyTweetIdSet(result.id)).take(numAdditionalReplies)

} else {

Seq.empty

}

}

val transformedSearchResults =

topResults ++ additionalReplies ++ randomSearchResults

.map(\_.\_1)

resultCounter.incr(transformedSearchResults.size)

additionalReplyCounter.incr(additionalReplies.size)

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

}

}