package com.twitter.home\_mixer.product.scored\_tweets.scorer

import com.twitter.home\_mixer.functional\_component.scorer.FeedbackFatigueScorer

import com.twitter.home\_mixer.model.HomeFeatures

import com.twitter.home\_mixer.model.HomeFeatures.AuthorIsBlueVerifiedFeature

import com.twitter.home\_mixer.model.HomeFeatures.AuthorIsCreatorFeature

import com.twitter.home\_mixer.model.HomeFeatures.FeedbackHistoryFeature

import com.twitter.home\_mixer.model.HomeFeatures.InNetworkFeature

import com.twitter.home\_mixer.model.HomeFeatures.InReplyToTweetIdFeature

import com.twitter.home\_mixer.product.scored\_tweets.param.ScoredTweetsParam.BlueVerifiedAuthorInNetworkMultiplierParam

import com.twitter.home\_mixer.product.scored\_tweets.param.ScoredTweetsParam.BlueVerifiedAuthorOutOfNetworkMultiplierParam

import com.twitter.home\_mixer.product.scored\_tweets.param.ScoredTweetsParam.CreatorInNetworkMultiplierParam

import com.twitter.home\_mixer.product.scored\_tweets.param.ScoredTweetsParam.CreatorOutOfNetworkMultiplierParam

import com.twitter.home\_mixer.product.scored\_tweets.param.ScoredTweetsParam.OutOfNetworkScaleFactorParam

import com.twitter.home\_mixer.util.CandidatesUtil

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.timelineservice.{thriftscala => tls}

trait RescoringFactorProvider {

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double

def apply(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate],

): Double = if (selector(candidate)) factor(query, candidate) else 1.0

}

/\*\*

\* Re-scoring multiplier to apply to authors who are eligible subscription content creators

\*/

object RescoreCreators extends RescoringFactorProvider {

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean =

candidate.features.getOrElse(AuthorIsCreatorFeature, false) &&

CandidatesUtil.isOriginalTweet(candidate)

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double =

if (candidate.features.getOrElse(InNetworkFeature, false))

query.params(CreatorInNetworkMultiplierParam)

else query.params(CreatorOutOfNetworkMultiplierParam)

}

/\*\*

\* Re-scoring multiplier to apply to authors who are verified by Twitter Blue

\*/

object RescoreBlueVerified extends RescoringFactorProvider {

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean =

candidate.features.getOrElse(AuthorIsBlueVerifiedFeature, false) &&

CandidatesUtil.isOriginalTweet(candidate)

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double =

if (candidate.features.getOrElse(InNetworkFeature, false))

query.params(BlueVerifiedAuthorInNetworkMultiplierParam)

else query.params(BlueVerifiedAuthorOutOfNetworkMultiplierParam)

}

/\*\*

\* Re-scoring multiplier to apply to out-of-network tweets

\*/

object RescoreOutOfNetwork extends RescoringFactorProvider {

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean =

!candidate.features.getOrElse(InNetworkFeature, false)

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double = query.params(OutOfNetworkScaleFactorParam)

}

/\*\*

\* Re-scoring multiplier to apply to reply candidates

\*/

object RescoreReplies extends RescoringFactorProvider {

private val ScaleFactor = 0.75

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean =

candidate.features.getOrElse(InReplyToTweetIdFeature, None).isDefined

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double = ScaleFactor

}

/\*\*

\* Re-scoring multiplier to calibrate multi-tasks learning model prediction

\*/

object RescoreMTLNormalization extends RescoringFactorProvider {

private val ScaleFactor = 1.0

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean = {

candidate.features.contains(HomeFeatures.FocalTweetAuthorIdFeature)

}

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double = ScaleFactor

}

/\*\*

\* Re-scoring multiplier to apply to multiple tweets from the same author

\*/

case class RescoreAuthorDiversity(diversityDiscounts: Map[Long, Double])

extends RescoringFactorProvider {

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean =

diversityDiscounts.contains(candidate.candidate.id)

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double = diversityDiscounts(candidate.candidate.id)

}

case class RescoreFeedbackFatigue(query: PipelineQuery) extends RescoringFactorProvider {

def selector(candidate: CandidateWithFeatures[TweetCandidate]): Boolean = true

private val feedbackEntriesByEngagementType =

query.features

.getOrElse(FeatureMap.empty).getOrElse(FeedbackHistoryFeature, Seq.empty)

.filter { entry =>

val timeSinceFeedback = query.queryTime.minus(entry.timestamp)

timeSinceFeedback < FeedbackFatigueScorer.DurationForFiltering + FeedbackFatigueScorer.DurationForDiscounting &&

entry.feedbackType == tls.FeedbackType.SeeFewer

}.groupBy(\_.engagementType)

private val authorsToDiscount =

FeedbackFatigueScorer.getUserDiscounts(

query.queryTime,

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Tweet, Seq.empty))

private val likersToDiscount =

FeedbackFatigueScorer.getUserDiscounts(

query.queryTime,

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Like, Seq.empty))

private val followersToDiscount =

FeedbackFatigueScorer.getUserDiscounts(

query.queryTime,

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Follow, Seq.empty))

private val retweetersToDiscount =

FeedbackFatigueScorer.getUserDiscounts(

query.queryTime,

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Retweet, Seq.empty))

def factor(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Double = {

FeedbackFatigueScorer.getScoreMultiplier(

candidate,

authorsToDiscount,

likersToDiscount,

followersToDiscount,

retweetersToDiscount

)

}

}