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

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

import com.twitter.home\_mixer.product.scored\_tweets.model.ScoredTweetsQuery

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

import com.twitter.product\_mixer.core.feature.Feature

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

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

import com.twitter.product\_mixer.core.functional\_component.scorer.Scorer

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

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

import com.twitter.stitch.Stitch

/\*\*

\* Apply various heuristics to the model score

\*/

object HeuristicScorer extends Scorer[ScoredTweetsQuery, TweetCandidate] {

override val identifier: ScorerIdentifier = ScorerIdentifier("Heuristic")

override val features: Set[Feature[\_, \_]] = Set(ScoreFeature)

override def apply(

query: ScoredTweetsQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = {

val rescorers = Seq(

RescoreOutOfNetwork,

RescoreReplies,

RescoreBlueVerified,

RescoreCreators,

RescoreMTLNormalization,

RescoreAuthorDiversity(AuthorDiversityDiscountProvider(candidates)),

RescoreFeedbackFatigue(query)

)

val updatedScores = candidates.map { candidate =>

val score = candidate.features.getOrElse(ScoreFeature, None)

val scaleFactor = rescorers.map(\_(query, candidate)).product

val updatedScore = score.map(\_ \* scaleFactor)

FeatureMapBuilder().add(ScoreFeature, updatedScore).build()

}

Stitch.value(updatedScores)

}

}