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

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

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

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

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

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

import com.twitter.product\_mixer.core.functional\_component.filter.Filter

import com.twitter.product\_mixer.core.functional\_component.filter.FilterResult

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

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

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

import com.twitter.stitch.Stitch

object OutOfNetworkCompetitorFilter extends Filter[PipelineQuery, TweetCandidate] {

override val identifier: FilterIdentifier = FilterIdentifier("OutOfNetworkCompetitor")

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val competitorAuthors = query.params(CompetitorSetParam)

val (removed, kept) =

candidates.partition(isOutOfNetworkTweetFromCompetitor(\_, competitorAuthors))

Stitch.value(FilterResult(kept = kept.map(\_.candidate), removed = removed.map(\_.candidate)))

}

def isOutOfNetworkTweetFromCompetitor(

candidate: CandidateWithFeatures[TweetCandidate],

competitorAuthors: Set[Long]

): Boolean = {

!candidate.features.getOrElse(InNetworkFeature, true) &&

!candidate.features.getOrElse(IsRetweetFeature, false) &&

candidate.features.getOrElse(AuthorIdFeature, None).exists(competitorAuthors.contains)

}

}