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

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

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

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

/\*\*

\* Remove any candidate that is in the ancestor list of any reply, including retweets of ancestors.

\*

\* E.g. if B replied to A and D was a retweet of A, we would prefer to drop D since otherwise

\* we may end up serving the same tweet twice in the timeline (e.g. serving both A->B and D).

\*/

object DuplicateConversationTweetsFilter extends Filter[PipelineQuery, TweetCandidate] {

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

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val allAncestors = candidates

.flatMap(\_.features.getOrElse(AncestorsFeature, Seq.empty))

.map(\_.tweetId).toSet

val (kept, removed) = candidates.partition { candidate =>

!allAncestors.contains(CandidatesUtil.getOriginalTweetId(candidate))

}

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

}

}