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

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

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

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

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

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.selector.SelectorResult

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

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

case class KeepBestOutOfNetworkCandidatePerAuthorPerSuggestType(

override val pipelineScope: CandidateScope)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val (selectedCandidates, otherCandidates) =

remainingCandidates.partition(candidate =>

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

val filteredCandidates = selectedCandidates

.groupBy { candidate =>

(

candidate.features.getOrElse(AuthorIdFeature, None),

candidate.features.getOrElse(SuggestTypeFeature, None)

)

}

.values.map(\_.maxBy(\_.features.getOrElse(ScoreFeature, None)))

.toSeq

val updatedCandidates = otherCandidates ++ filteredCandidates

SelectorResult(remainingCandidates = updatedCandidates, result = result)

}

}