package com.twitter.product\_mixer.component\_library.filter

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.UniversalNoun

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

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

import com.twitter.stitch.Stitch

import com.twitter.search.common.util.bloomfilter.AdaptiveLongIntBloomFilter

trait GetAdaptiveLongIntBloomFilter[Query <: PipelineQuery] {

def apply(query: Query): Option[AdaptiveLongIntBloomFilter]

}

case class AdaptiveLongIntBloomFilterDedupFilter[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Long]

](

getBloomFilter: GetAdaptiveLongIntBloomFilter[Query])

extends Filter[Query, Candidate] {

override val identifier: FilterIdentifier = FilterIdentifier(

"AdaptiveLongIntBloomFilterDedupFilter")

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[FilterResult[Candidate]] = {

val filterResult = getBloomFilter(query)

.map { bloomFilter =>

val (kept, removed) =

candidates.map(\_.candidate).partition(candidate => !bloomFilter.contains(candidate.id))

FilterResult(kept, removed)

}.getOrElse(FilterResult(candidates.map(\_.candidate), Seq.empty))

Stitch.value(filterResult)

}

}