package com.twitter.follow\_recommendations.common.candidate\_sources.sims

import com.google.inject.Singleton

import com.twitter.follow\_recommendations.common.candidate\_sources.sims.Follow2vecNearestNeighborsStore.NearestNeighborParamsType

import com.twitter.hermit.candidate.thriftscala.Candidate

import com.twitter.hermit.candidate.thriftscala.Candidates

import com.twitter.hermit.model.Algorithm

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

import com.twitter.stitch.Stitch

import com.twitter.strato.catalog.Fetch

import com.twitter.strato.client.Fetcher

import com.twitter.strato.generated.client.recommendations.follow2vec.LinearRegressionFollow2vecNearestNeighborsClientColumn

import com.twitter.util.Return

import com.twitter.util.Throw

import javax.inject.Inject

@Singleton

class LinearRegressionFollow2vecNearestNeighborsStore @Inject() (

linearRegressionFollow2vecNearestNeighborsClientColumn: LinearRegressionFollow2vecNearestNeighborsClientColumn)

extends StratoBasedSimsCandidateSource[NearestNeighborParamsType](

Follow2vecNearestNeighborsStore.convertFetcher(

linearRegressionFollow2vecNearestNeighborsClientColumn.fetcher),

view = Follow2vecNearestNeighborsStore.defaultSearchParams,

identifier = Follow2vecNearestNeighborsStore.IdentifierF2vLinearRegression

)

object Follow2vecNearestNeighborsStore {

// (userid, feature store version for data)

type NearestNeighborKeyType = (Long, Long)

// (neighbors to be returned, ef value: accuracy / latency tradeoff, distance for filtering)

type NearestNeighborParamsType = (Option[Int], Option[Int], Option[Double])

// (seq(found neighbor id, score), distance for filtering)

type NearestNeighborValueType = (Seq[(Long, Option[Double])], Option[Double])

val IdentifierF2vLinearRegression: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.LinearRegressionFollow2VecNearestNeighbors.toString)

val defaultFeatureStoreVersion: Long = 20210708

val defaultSearchParams: NearestNeighborParamsType = (None, None, None)

def convertFetcher(

fetcher: Fetcher[NearestNeighborKeyType, NearestNeighborParamsType, NearestNeighborValueType]

): Fetcher[Long, NearestNeighborParamsType, Candidates] = {

(key: Long, view: NearestNeighborParamsType) =>

{

def toCandidates(

results: Option[NearestNeighborValueType]

): Option[Candidates] = {

results.flatMap { r =>

Some(

Candidates(

key,

r.\_1.map { neighbor =>

Candidate(neighbor.\_1, neighbor.\_2.getOrElse(0))

}

)

)

}

}

val results: Stitch[Fetch.Result[NearestNeighborValueType]] =

fetcher.fetch(key = (key, defaultFeatureStoreVersion), view = view)

results.transform {

case Return(r) => Stitch.value(Fetch.Result(toCandidates(r.v)))

case Throw(e) => Stitch.exception(e)

}

}

}

}