package com.twitter.simclusters\_v2.candidate\_source

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.Stats

import com.twitter.simclusters\_v2.candidate\_source.SimClustersANNCandidateSource.SimClustersTweetCandidate

import com.twitter.simclusters\_v2.thriftscala.EmbeddingType

import com.twitter.simclusters\_v2.thriftscala.InternalId

import com.twitter.simclusters\_v2.thriftscala.ScoreInternalId

import com.twitter.simclusters\_v2.thriftscala.ScoringAlgorithm

import com.twitter.simclusters\_v2.thriftscala.SimClustersEmbeddingId

import com.twitter.simclusters\_v2.thriftscala.SimClustersEmbeddingPairScoreId

import com.twitter.simclusters\_v2.thriftscala.{Score => ThriftScore}

import com.twitter.simclusters\_v2.thriftscala.{ScoreId => ThriftScoreId}

import com.twitter.util.Future

import com.twitter.storehaus.ReadableStore

object HeavyRanker {

trait HeavyRanker {

def rank(

scoringAlgorithm: ScoringAlgorithm,

sourceEmbeddingId: SimClustersEmbeddingId,

candidateEmbeddingType: EmbeddingType,

minScore: Double,

candidates: Seq[SimClustersTweetCandidate]

): Future[Seq[SimClustersTweetCandidate]]

}

class UniformScoreStoreRanker(

uniformScoringStore: ReadableStore[ThriftScoreId, ThriftScore],

stats: StatsReceiver)

extends HeavyRanker {

val fetchCandidateEmbeddingsStat = stats.scope("fetchCandidateEmbeddings")

def rank(

scoringAlgorithm: ScoringAlgorithm,

sourceEmbeddingId: SimClustersEmbeddingId,

candidateEmbeddingType: EmbeddingType,

minScore: Double,

candidates: Seq[SimClustersTweetCandidate]

): Future[Seq[SimClustersTweetCandidate]] = {

val pairScoreIds = candidates.map { candidate =>

ThriftScoreId(

scoringAlgorithm,

ScoreInternalId.SimClustersEmbeddingPairScoreId(

SimClustersEmbeddingPairScoreId(

sourceEmbeddingId,

SimClustersEmbeddingId(

candidateEmbeddingType,

sourceEmbeddingId.modelVersion,

InternalId.TweetId(candidate.tweetId)

)

))

) -> candidate.tweetId

}.toMap

Future

.collect {

Stats.trackMap(fetchCandidateEmbeddingsStat) {

uniformScoringStore.multiGet(pairScoreIds.keySet)

}

}

.map { candidateScores =>

candidateScores.toSeq

.collect {

case (pairScoreId, Some(score)) if score.score >= minScore =>

SimClustersTweetCandidate(pairScoreIds(pairScoreId), score.score, sourceEmbeddingId)

}

}

}

}

}