package com.twitter.cr\_mixer.similarity\_engine

import com.twitter.cr\_mixer.config.SimClustersANNConfig

import com.twitter.cr\_mixer.model.SimilarityEngineInfo

import com.twitter.cr\_mixer.model.TweetWithScore

import com.twitter.cr\_mixer.thriftscala.SimilarityEngineType

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.util.StatsUtil

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

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

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

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

import com.twitter.simclustersann.thriftscala.SimClustersANNService

import com.twitter.simclustersann.thriftscala.{Query => SimClustersANNQuery}

import com.twitter.storehaus.ReadableStore

import com.twitter.timelines.configapi

import com.twitter.util.Future

import javax.inject.Singleton

import com.twitter.cr\_mixer.exception.InvalidSANNConfigException

import com.twitter.relevance\_platform.simclustersann.multicluster.ServiceNameMapper

@Singleton

case class SimClustersANNSimilarityEngine(

simClustersANNServiceNameToClientMapper: Map[String, SimClustersANNService.MethodPerEndpoint],

statsReceiver: StatsReceiver)

extends ReadableStore[

SimClustersANNSimilarityEngine.Query,

Seq[TweetWithScore]

] {

private val name: String = this.getClass.getSimpleName

private val stats = statsReceiver.scope(name)

private val fetchCandidatesStat = stats.scope("fetchCandidates")

private def getSimClustersANNService(

query: SimClustersANNQuery

): Option[SimClustersANNService.MethodPerEndpoint] = {

ServiceNameMapper

.getServiceName(

query.sourceEmbeddingId.modelVersion,

query.config.candidateEmbeddingType).flatMap(serviceName =>

simClustersANNServiceNameToClientMapper.get(serviceName))

}

override def get(

query: SimClustersANNSimilarityEngine.Query

): Future[Option[Seq[TweetWithScore]]] = {

StatsUtil.trackOptionItemsStats(fetchCandidatesStat) {

getSimClustersANNService(query.simClustersANNQuery) match {

case Some(simClustersANNService) =>

simClustersANNService.getTweetCandidates(query.simClustersANNQuery).map {

simClustersANNTweetCandidates =>

val tweetWithScores = simClustersANNTweetCandidates.map { candidate =>

TweetWithScore(candidate.tweetId, candidate.score)

}

Some(tweetWithScores)

}

case None =>

throw InvalidSANNConfigException(

"No SANN Cluster configured to serve this query, check CandidateEmbeddingType and ModelVersion")

}

}

}

}

object SimClustersANNSimilarityEngine {

case class Query(

simClustersANNQuery: SimClustersANNQuery,

simClustersANNConfigId: String)

def toSimilarityEngineInfo(

query: EngineQuery[Query],

score: Double

): SimilarityEngineInfo = {

SimilarityEngineInfo(

similarityEngineType = SimilarityEngineType.SimClustersANN,

modelId = Some(

s"SimClustersANN\_${query.storeQuery.simClustersANNQuery.sourceEmbeddingId.embeddingType.toString}\_" +

s"${query.storeQuery.simClustersANNQuery.sourceEmbeddingId.modelVersion.toString}\_" +

s"${query.storeQuery.simClustersANNConfigId}"),

score = Some(score)

)

}

def fromParams(

internalId: InternalId,

embeddingType: EmbeddingType,

modelVersion: ModelVersion,

simClustersANNConfigId: String,

params: configapi.Params,

): EngineQuery[Query] = {

// SimClusters EmbeddingId and ANNConfig

val simClustersEmbeddingId =

SimClustersEmbeddingId(embeddingType, modelVersion, internalId)

val simClustersANNConfig =

SimClustersANNConfig

.getConfig(embeddingType.toString, modelVersion.toString, simClustersANNConfigId)

EngineQuery(

Query(

SimClustersANNQuery(

sourceEmbeddingId = simClustersEmbeddingId,

config = simClustersANNConfig.toSANNConfigThrift

),

simClustersANNConfigId

),

params

)

}

}